PROGRAMAR

REVISTA PORTUGUESA DE PROGRAMAÇÃO • WWW.PORTUGAL-A-PROGRAMAR.ORG

EDIÇÃO #31 - OUTUBRO 2011

ISSN 1647-0710

NHIBERNATE

TÉCNICAS PARA SOFTWARE MELHOR E MAIS RÁPIDO

EDITORIAL

ISSN
1 647-071 0

(Des)Informação

Actualmente muito se têm falado sobre tudo. Falado e especulado. Com a saída de Steve Jobs da Apple especula-se qual serão as implicações para a marca da maçã. Com o lançamento do Windows 8 especula-se sobre a capacidade deste conquistar o mercado dos tablets. Com a conferência da Apple "Let's talk about IPhone" especulou-se sobre o possível anúncio da data de lançamento do IPhone 5. Com o abandono da Nokia, especulou-se o que aconteceria ao sistema operativo Meego. E muito mais se especula.

Infelizmente a nossa sociedade vive presa às especulações, e creio que o melhor exemplo disso, nem está na área tecnológica, mas sim na área financeira com a crise em que vivemos ser altamente ampliada pelas especulações. Há uns anos atrás sempre que alguém queria dar um exemplo financeiramente forte sobre especulações falava sobre a famosa Quinta-feira negra, onde as acções de um dia tiveram efeitos nos anos seguintes e onde aconteceu a muitos acordar rico e adormecer pobre. Todavia hoje não precisámos de dar um exemplo tão longínquo no tempo e em que muitos não conhecem as verdadeiras consequências. Agora podemos fala no presente.

Mas não precisámos de falar apenas na área financeira, podemos falar de qualquer área, inclusivamente a informática. Apesar de o IPhone 5 ainda não ter sido lançado, nem existirem provas de que alguma vez irá ver a luz do dia, a "publicidade" feita em torno dele, torna-o no alvo de curiosidade para o bem e para o mal. Chamar-lhe-ia publicidade gratuita. Se eu dissesse que era provável um canal de televisão deixasse de existir no dia x, estaria a especular. Provavelmente teria pouco reflexo na sociedade, uma vez que os próprios meios de comunicação social me dariam pouco tempo de antena, ao invés destes assuntos sobre grandes empresas. Contudo estaria a distorcer o mercado, criando expectativa em volta de algo que eu apenas defendia/acreditava, mas sem ter provas ou factos.

Numa área cujo nome deriva de Informação automática, a verdadeira informação que é processada, comparativamente com o total de dados é cada vez menor. Cada vez mais recorremos a uma ferramenta que deveria armazenar e processar dados gerando informação, contudo devido à falta de fidedignidade das fontes e contrariedade dos dados lá introduzidos é muitas vezes gerada desinformação. Se eu disse-se que o mundo acabava hoje dentro de 5 minutos, mesmo que muita gente acredita-se em mim pouco podia fazer até verificar a veracidade dessa informação. Passados 5 minutos saberiam se era verdade ou não. Se eu disser que o mundo acaba em 21 de Dezembro de 2012 ainda se terá muito que esperar até se conseguir confirmar ou não a afirmação. Por isso não é informação, mas sim especulação. Especulação que faz vender, que mexe com os mercados, que coloca em muitos os nervos à flor da pele, que deixa outros com um sorriso de vitória, que impede o sono tranquilo de muitos, mas que não passa de uma simples afirmação, cuja veracidade não pode ser demonstrada na altura em que foi proferida.

NR: Em notícia de última hora soube-se do falecimento de Steve Jobs. Para o bem e para o mal foi um nome incontornável no mundo da tecnologia, e que contribuiu para a sua evolução., por isso aqui ficam registadas as nossas condolências.

António Silva <antonio.silva@revista-programar.info>

TEMA DE CAPA

A PROGRAMAR

COLUNAS

COMUNIDADES

EVENTOS

19 a 21 Out	**Capsi 2011**
18 Out	**Upload Lisboa**
28 Out	**24ª Reunião Presencial NetPonto**
10 a 12 Nov	**Sapo Codebits V**
02 Nov	**SASIG 4**

Para mais informações/eventos: http://bit.ly/PAP_Eventos

O melhor amigo do Ipad é português e ecológico

"Kork2 foi desenhado pela A Produkt e produzido em Paços de Brandão num material reciclado e reciclável.

É o "canivete suíço" do Ipad e foi desenhado pela A Produkt, empresa sediada em Portugal rendida aos encantos da cortiça. O Kork2 – sucessor do Kork por direito – ajusta-se ao Ipad2, protegendo-o e tornando a sua utilização ainda mais confortável e dinâmica.

Simples, leve, ergonómico, recortado (pela 3DCork, de Paços de Brandão) num material reciclado e reciclável. São estes alguns dos segredos que atraíram o Kork2 até às prateleiras do MoMA de Nova Iorque e de Tóquio.

Todos os pormenores são estudados ao milímetro por Michael Andersen e Benedita Feijó, dupla que se lançou no mercado Apple com o StickyStrap para o iPhone antes de colocarem o iPad nas passarelas internacionais.

"Fazemos um exercício de especulação porque os desenhos da Apple estão normalmente fechados a sete chaves", explicou ao P3 Michael Andersen, ansioso pelos desenhos detalhados do iPhone5 e do iPad3.

"Vamos trabalhando com esses rumores até termos o objecto nas mãos para avançarmos com os desenhos finais. Os moldes são feitos ao milímetro porque a cortiça é um material extremamente complicado de moldar", prossegue um dos fundadores da A Produkt, responsável por "produtos originais" que, graças a "materiais portugueses", fazem a "ponte entre a tecnologia e a natureza". A A Produkt não exclui uma futura aventura no mundo da cerâmica e da indústria têxtil.

Fonte: Público, texto de Luís Octávio Costa

Windows 8 - Novo Explorador

Segundo o blog oficial da equipa de desenvolvimento do Windows 8, o explorador do Windows à semelhança do Microsoft Office, uma ribbon. Outra das novidades, é o facto deste possibilitar a abertura de ficheiros iso e vhd sem necessidade de aplicações de terceiros.

Imagens do novo explorador neste site: http://bit.ly/rj8vjl
Blog oficial do Windows 8: http://blogs.msdn.com/b/b8/

Fonte: P@P

Apple pode continuar a limitar uso do Mac OS aos seus computadores

A Apple tem o direito de restringir a utilização do sistema operativo OS X aos computadores fabricados pela empresa. A decisão do tribunal de recurso norte-americano foi conhecida esta semana, três anos depois de iniciada a batalha legal contra a Psystar, uma fabricante de "clones" dos produtos da maçã. A sentença anterior já havia dado razão à empresa de Steve Jobs, mas a Psystar - embora não tenha reclamado da violação de direitos de autor de que foi acusada por copiar o hardware da Apple - interpôs recurso da decisão alegando que as condições de licenciamento impostas para o Mac OS constituíam uma tentativa ilegal para estender a proteção dos direitos de autor a produtos que não podem beneficiar dela.

A Psystar alegou que ao limitar o uso do software da Apple aos produtos fabricados pela marca, a empresa estava a recorrer indevidamente à proteção que a lei dispensa aos direitos de autor para falsear a concorrência. Note-se que a fabricante de "sucedâneos" dos Mac tinha comprado cópias do Mac OS X para instalar nos computadores que vendia.

A juíza considerou que a concorrente não tinha apresentado provas de que as condições de licenciamento do sistema operativo fossem lesivas da concorrência ou criatividade. Os termos das licenças do software da Apple "não restringem a possibilidade de os concorrentes desenvolverem o seu próprio software, nem impede os clientes de usarem componentes de outra marca em computadores Apple", entendeu a magistrada, citada pela Ars Technica.

O que a Apple faz é restringir o uso de software da marca ao seu próprio hardware e a Psystar, que produz os seus próprios computadores, é livre de desenvolver software para eles, concluiu.

A decisão chega no culminar de um processo que teve início em Julho de 2008, quando a Apple pediu a condenação da Psystar por copiar os seus produtos. A sentença representa uma vitória importante para a fabricante do iPhone, que vê corroborada a "intocabilidade" do seu ecossistema e assegurado o direito a não disponibilizar o seu sistema operativo a outras fabricantes.

Escrito ao abrigo do novo Acordo Ortográfico

Fonte: Tek Sapo

Investigadores quebram criptografia SSL

Acesso a bancos portugueses e estrangeiros em risco

A segurança perfeita é um mito…já dizia o ditado! Nos dias de hoje, a segurança de muitos sites (especialmente de home-banking, ou compras online) passa pelo uso do protocolo SSL (Secure Sockets Layer), que permite (permitia) a integridade e confidencialidade dos dados que são passados entre o cliente e o servidor. Segundo informações disponíveis no site *The Register*, os investigadores tailandeses Thai Duong e Juliano Rizzo conseguiram pela primeira vez, quebrar o, até agora inquebrável, protocolo SSL e assim decifrar a informação que é passada ponto a ponto (entre o cliente e o servidor).

A noticia é preocupante e alarmante mas a descoberta levada a cabo pelo grupo de investigadores apenas põe em causa a segurança as versões 1.0 e anteriores do TLS (Transport Layer Security) – o sucessor do SSL. Até ao momento não há qualquer informação sobre vulnerabilidades que afectem a versão 1.1 ou 1.2 do TLS.

A demonstração de tal feito ocorrerá na conferência Ekoparty , que se realizará em Buenos Aires no final desta semana. Thai Duong e Juliano Rizzo irão apresentar o exploit que usaram para conseguir derrubar o SSL e ao qual deram o nome de BEAST. O exploit foi programado em JavaScript e funciona como um snifer, tendo a capacidade de decifrar a informação. Segundo os autores, o exploit funciona também em sites que usam HSTS (HTTP Strict Transport Security). Duong referiu que a demo que irão apresentar na conferência Ekoparty, consistirá na decifragem de um cookie de autenticação, utilizado no serviço Paypal.

Mas o que é o SSL? SSL é um protocolo criptográfico baseado em cifras assimétricas que providencia segurança e integridade dos dados transmitidos em redes como a Internet. Este protocolo ajuda a prevenir que entre as duas extremidades da comunicação não exista nenhum acesso indevido ou falsifiquem os dados transmitidos, ou seja, apenas o emissor e o receptor podem ver a informação da mensagem transmitida. Existem várias aplicações para este protocolo, como por exemplo o comércio electrónico, servidores Web, servidores FTP, etc.Para identificar facilmente se estão a visualizar um site seguro basta verificar no URL que em vez de estar o normal http:// se encontra https://. Para saber mais: http://bit.ly/p1MV0Y

Fonte: Sapo

BerliOS será fechado em 31.12.2011

O BerliOS foi fundado há 10 anos como um dos primeiros depósitos na Europa. Foi desenvolvido e mantido por Fraunhofer FOKUS. Como Europeu, o projecto não-proprietário BerliOS tinha como objective apoiar vários intervenientes e providenciar uma função de mediação neutra. Em 2011 cerca de 4710 projectos foram alojados no BerliOS, com 50,000 utilizadores registados e mais de 2.6 milhões de ficheiros descarregados cada mês. Estamos orgulhosos de, com o BerliOS, termos trazido a ideia de um depósito OSS à Europa. Entretanto, o conceito tem permanecido e existem muitas alternativas boas.

Infelizmente, como instituto de pesquisa Fraunhofer FOKUS tem apenas algumas oportunidades para operar um depósito como o BerliOS. Tal projecto apenas funcionará com um financiamento de seguimento, ou com patrocinadores ou parceiros que assumam o controlo do depósito. No campo de OSS é um empreendimento difícil. Num inquérito recente a comunidade indicou algum apoio em fundos e mão-de-obra pelos quais gostaríamos de vos agradecer. Infelizmente, o resultado não é suficiente para colocar o projecto numa base financeira sustentável. Além disso, a procura de patrocinadores ou parceiros foi sem êxito.

O Open Source é entendido pela Fraunhofer FOKUS como um paradigma para a futura utilização de inteligência orientada da TI. Magoa-nos ainda mais o facto de sermos forçados a descontinuar o alojamento para BerliOS em 31/12/2011.

Como programador, deverá exportar o seu projecto BerliOS para outro depósito. Como alternativas ver: http://bit.ly/18Pp6N . No nosso site irão encontrar um guia em como retirar os dados dos vossos projectos do portal e migrá-los para uma plataforma diferente. Ver: http://bit.ly/qGJfAo .

A Fraunhofer FOKUS tem um forte compromisso com o Open Source e interoperabilidade, e está envolvida em numerosos projectos OSS de sucesso. O instituto está focado no desenvolvimento de padrões de qualidade para o software open source e em particular na inoperabilidade técnica, semântica e organizacional entre componentes de software open source e software closed source. Exemplo das nossas actividade OSS incluindo a nossa gestão do Centro de Competência Alemão QualiPSo.

Agradecemos a todos terem utilizado o BerliOS ao longo dos anos.Fraunhofer FOKUS http://www.fokus.fraunhofer.de

Fonte: www.berlios.de

Tradução: Sara Santos

TEMA DA CAPA

NHibernate - Técnicas para software melhor e mais rápido

NHibernate - Técnicas para software melhor e mais rápido

No artigo da edição anterior abordamos os temas de configuração de NHibernate, diferentes formas de mapear as entidades e diferentes formas de obter os dados da base de dados no contexto do seguinte domínio:

```
public class Edicao
{
  public virtual DateTime DataEdicao {get; set; }
  public virtual Artigo TemaDeCapa { get; set; }
  public virtual IList<Artigo> Artigos {get;set;}
}

public class Artigo
{
  public virtual int Id { get; set; }
  public virtual string Titulo { get; set; }
  public virtual ISet<Autor> Autores {get;set; }
}

public class Autor
{
  public virtual int Id { get; set; }
  public virtual string Nome { get; set; }
  public virtual string Email { get; set; }
  public virtual ISet<Artigo> Artigos {get;set; }
}
```

Neste artigo vamos ver como podemos aproveitar algumas das funcionalidades mais avançadas do NHibernate para tornar as nossas aplicações mais eficientes, reduzindo o número de pedidos à base de dados assim como a quantidade de dados que são transmitidos. Em certos momentos iremos usar a ferramenta NHProf para observar a comunicação entre a nossa aplicação e a base de dados.

Performance

Uma das falácias de computação distribuída é de que latência é zero. Na realidade, qualquer chamada a um sistema remoto é ordens de magnitude mais lenta do que a mesma chamada a um sistema local e dentro do mesmo processo. Outra falácia é de que a largura de banda é infinita. Como tal é boa prática reduzir o número de chamadas remotas ao servidor de dados e o tamanho dos dados a receber e enviar se queremos ter aplicações com boa performance mesmo quando em carga.

Como reduzir o número de chamadas

Um dos casos típicos onde fazemos mais chamadas à base de dados do que o necessário é quando mostramos uma listagem de itens que necessita de dados de mais do que uma tabela:

```
foreach (var artigo in
      session.QueryOver<Artigo>().List())
{
  var autores = artigo.Autores.Select(a =>
              a.Nome);

    Console.Out.WriteLine(
            "Artigo {0} com autores {1}",
            artigo.Titulo,
            string.Join(", ", autores.ToArray()));
}
```

Neste caso, estamos a mostrar o título de um artigo em conjunto com os autores. Se olharmos para o SQL gerado vemos que por cada artigo vamos à base de dados buscar os autores:

Short SQL	Row Count
begin transaction with isolation level: Unspecified	
SELECT ... FROM [Artigo] this_	4
SELECT ... FROM ArtigosToAutores autores0_ left outer join [Aut... WHERE autores0_.Artigo_id = 4242	1
SELECT ... FROM ArtigosToAutores autores0_ left outer join [Aut... WHERE autores0_.Artigo_id = 4243	1
SELECT ... FROM ArtigosToAutores autores0_ left outer join [Aut... WHERE autores0_.Artigo_id = 4244	2
SELECT ... FROM ArtigosToAutores autores0_ left outer join [Aut... WHERE autores0_.Artigo_id = 4245	0
commit transaction	

Este comportamento é resultante das capacidades de lazy-loading da biblioteca em conjunto com as configurações por omissão que carregam os objectos de cada relação apenas quando esta é acedida.

Tal comportamento implica que vão ser efectuados no mínimo tantos queries quanto artigos mostrarmos. Qualquer latência no acesso ao servidor de sql aumenta significativamente o tempo de resposta da nossa aplicação, quando na realidade seria possível minimizar as chamadas à base de dados e obter todos os dados de uma só vez.

Select N+1

A esta classe de problemas dá-se o nome de "Select N+1", em virtude de ser efectuada uma chamada à base de dados para obter a lista de objectos raiz, e posteriormente N chamadas para obter os objectos relacionados de cada raiz. É uma fonte típica de problemas de performance, e em quase todos os casos pode ser resolvida com um carregamento mais inteligente dos dados.

Eager-Loading

Embora a intuição seja de que o oposto de lazy-loading seja eager-loading, na realidade existem várias formas de fazer eager-loading de dados, pelo que devem ser usadas em circunstâncias diferentes.

NHibernate - Técnicas para software melhor e mais rápido

É possível configurar este comportamento a nível dos mapeamentos globais ou durante um query à base de dados. Quando é configurado no mapeamento pode ser feito a uma classe ou a uma relação. Quando é feito durante o query é sempre respectivo a uma relação.

Quando uma classe é indicada como não lazy-loaded, qualquer instância da mesma criada pelo NHibernate deixará de ser uma proxy.

A implicação é de que qualquer relação que seja de um para um ou de muitos para um com uma classe que não possa ser lazy-loaded terá de ser logo concretizada.

Considerando que queremos que a classe **Artigo** não seja lazy-loaded:

```
public class ArtigoOverride : IAutoMappingOverri-
de<Artigo>
{
 public void Override(AutoMapping<Artigo>mapping)
    {
        mapping.Not.LazyLoad();
    }
}
```

Quando carregamos uma **Edicao** (que tem uma colecção de artigos e um tema de capa):

```
var edicao = session.Get<Edicao>(dataEdicao);
```

O sql gerado inclui já o join necessário para carregar o tema de capa:

```
SELECT
      edicao0_.DataEdicao    as DataEdicao3_1_,
      edicao0_.TemaDeCapa_id as TemaDeCapa2_3_1_,
      artigo1_.Id            as Id0_0_,
      artigo1_.Titulo        as Titulo0_0_
FROM  [Edicao] edicao0_
       left outer join [Artigo] artigo1_
         on edicao0_.TemaDeCapa_id = artigo1_.Id
WHERE
   edicao0_.DataEdicao = '2011-07-01T00:00:00.00'
```

Há aqui dois comportamentos que são necessário distinguir: o da relação de muitos para um (**TemaDeCapa**) e o da relação de um para muitos (**Artigos**).

Como é a classe **Artigo** que deixa de ser lazy-loaded, quando o NHibernate faz um query a partir da **Edicao** repara que não vai poder criar a proxy para o **TemaDeCapa**, e cria o sql necessário para carregar os dados do **Artigo** correspondente.

No caso dos **Artigos** não surge esse problema porque não necessita de criar as instâncias imediatamente, apenas uma lista que as vai guardar. Até iterarmos por esta lista não precisamos de construir as respectivas instâncias de **Artigos**.

Para carregar os dados dos **Artigos** sempre que uma edição é carregada, usamos a seguinte configuração:

```
public class EdicaoOverride : IAutoMappingOverri-
de<Edicao>
{
 public void Override(AutoMapping<Edicao>mapping)
    {
        mapping.HasMany(e => e.Artigos)
            .Not.LazyLoad();
    }
}
```

Neste caso estamos a indicar que quando uma **Edicao** é carregada, os **Artigos** também o são. A forma de carregar os dados varia entre três opções: select (comportamento por omissão), subselect e join.

Quando é usado o select, por cada entidade com a relação é efectuado um select à base de dados. Este é o comportamento que existe por omissão e quer dizer que continuamos com um problema de Select N+1:

Quando é usado o subselect, por cada query ou entidade obtida da base de dados é efectuado outro query que obtém os dados necessários para popular as relações. Este comportamento é interessante nas relações de muitos para muitos, como a entre **Artigos** e **Autores**. Se nós tivermos o seguinte query:

```
Foreach (
    var artigo in session.QueryOver<Artigo>()
      .Where(a => a.Id < idQuartoArtigo).List())
{
     var autores = artigo.Autores.Select(a =>
                 a.Nome);

     Console.Out.WriteLine(
            "Artigo {0} com autores {1}",
            artigo.Titulo,
            string.Join(", ", autores.ToArray()));
}
```

Os queries gerados por omissão são 4:

Short SQL	Row Count
begin transaction with isolation level: Unspecified	
SELECT ... FROM [Artigo] this_ WHERE this_.Id < 1316	3
SELECT ... FROM [Autor] autores0_ WHERE autores0_.Artigo_id = 1313	0
SELECT ... FROM [Autor] autores0_ WHERE autores0_.Artigo_id = 1314	0
SELECT ... FROM [Autor] autores0_ WHERE autores0_.Artigo_id = 1315	2
commit transaction	

Um para obter os artigos, e um por cada artigo.

No entanto, se configurarmos a relação com um fetch por subselect passamos a ter apenas dois queries:

Short SQL	Row Count
begin transaction with isolation level: Unspecified	
SELECT ... FROM [Artigo] this_ WHERE this_.Id < 1922	3
SELECT ... FROM [Autor] autores0_ WHERE autores0_.Artigo_id in (Select this_.Id From [A...	2 / 2
commit transaction	

Neste caso o select que popula os autores é filtrado pelo primeiro query através de um subselect:

```
SELECT autores0_.Artigo_id as Artigo4_1_,
       autores0_.Id        as Id1_,
       autores0_.Id        as Id1_0_,
       autores0_.Nome      as Nome1_0_,
       autores0_.Email     as Email1_0_
FROM   [Autor] autores0_
WHERE  autores0_.Artigo_id in
        (SELECT this_.Id
         FROM [Artigo] this_
         WHERE  this_.Id < 1922 /* @p0 */)
```

Reduz o número de queries efectuadas á base de dados, no entanto necessita que a base de dados suporte este mecanismo.

Quando é usado o join, como o nome indica, os dados da colecção são obtidos ao mesmo tempo que as entidades através de um join. Para o código mencionado anteriormente passamos a ter apenas um query:

```
SELECT this_.Id               as Id0_1_,
       this_.Titulo           as Titulo0_1_,
       autores2_.Artigo_id as Artigo4_3_,
       autores2_.Id           as Id3_,
       autores2_.Id           as Id1_0_,
       autores2_.Nome         as Nome1_0_,
       autores2_.Email        as Email1_0_
FROM   [Artigo] this_
       left outer join [Autor] autores2_
         on this_.Id = autores2_.Artigo_id
WHERE  this_.Id < 2225 /* @p0 */
```

É necessário ter em atenção a forma como se usa estas funcionalidades gerais de eager-loading. Por exemplo, ao configurar a colecção de Artigos de uma Edicao para ser automaticamente obtida por join, o seguinte código pode ter um resultado inesperado:

```
foreach (var edicao in session.QueryOver<Edicao>
().Take(2).List())
{
        Console.Out.WriteLine("Edicao {0} ",
                              edicao.DataEdicao);
}
```

À primeira vista deveríamos obter duas edições distintas. E em muitos casos é isso que temos. No entanto ocasionalmente encontramos edições duplicadas, e ao olharmos para o sql gerado vemos qual o problema:

```
SELECT TOP ( 2 /* @p0 */ )
      this_.DataEdicao    as DataEdicao3_1_,
      this_.TemaDeCapa_id as TemaDeCapa2_3_1_,
      artigos2_.Edicao_id as Edicao3_3_,
      artigos2_.Id        as Id3_,
      artigos2_.Id        as Id0_0_,
      artigos2_.Titulo    as Titulo0_0_
FROM [Edicao] this_
       left outer join [Artigo] artigos2_
        on this_.DataEdicao = artigos2_.Edicao_id
```

Neste caso como é feito o carregamento imediato dos artigos através de um join, vamos ter edições repetidas quando uma edição tem mais que um artigo. Como tal, esta opção não é aconselhada na maioria dos casos, sendo preferível configurar o carregamento de dados por query em vez de forma geral.

Configurado por relação no mapeamento tem também a desvantagem de carregar os dados mesmo que não sejam usados. Como tal, estas soluções são pouco flexíveis, visto assumirem que se pretende sempre os mesmos dados relacionados com as entidades em todos os casos de uso.

Para maior flexibilidade a solução acaba por passar por indicar no query quais as colecções que se pretende carregar imediatamente:

```
foreach (var artigo in session.QueryOver<Artigo>
().Fetch(a => a.Autores).Eager.List())
{
   Console.Out.WriteLine(
     "Artigo {0} com autores {1}",
     artigo.Titulo,
     string.Join(", ",
              artigo.Autores.Select(
              a => a.Nome).ToArray()));
}
```

Aqui estamos a indicar que queremos carregar a colecção Autores do Artigo juntamente com o artigo. O sql gerado usa joins para obter os dados das entidades relacionadas:

```
SELECT this_.Id            as Id0_1_,
       this_.Titulo        as Titulo0_1_,
       autores2_.Artigo_id as Artigo4_3_,
       autores2_.Id        as Id3_,
       autores2_.Id        as Id1_0_,
       autores2_.Nome      as Nome1_0_,
       autores2_.Email     as Email1_0_
FROM   [Artigo] this_
       left outer join [Autor] autores2_
         on this_.Id = autores2_.Artigo_id
```

Se olharmos para os métodos Get e Load, não temos nenhuma forma de configurar eager-loading nesses casos.

Como tal, a forma de o fazer é com uma query por id, configurando ai que relações é que pretendemos carregar logo:

```
foreach (var artigo in session.QueryOver<Artigo>
()
    .Where(a => a.Id == idTerceiroArtigo)
    .Fetch(a => a.Autores).Eager.List())
{
    var autores = artigo.Autores.Select(
                       a => a.Nome).ToArray();
    Console.Out.WriteLine(
        "Artigo {0} com autores {1}",
        artigo.Titulo,
        string.Join(", ", autores));
}
```

Neste caso também podemos vir a ter o problema de entidades duplicadas. No entanto como o carregamento é feito por query, podemos fazer o tuning em cada caso de uso, de acordo com as necessidades reais.

Mais à frente neste artigo vamos ver uma forma mais correcta e menos dada a erros de resolver este problema, fazendo o carregamento de dados de forma fácil e rápida.

Batch fetching

Outra forma de reduzir o número de vezes que efectuamos chamadas à base de dados é configurar relações com batch-fetching. Esta é uma configuração que apenas pode ser feita no mapeamento, por oposição a eager-loading que também pode ser feita por query.

Nestes casos, quando NHibernate detecta um acesso à colecção, ele procura mais entidades dentro da mesma sessão que tenham essa relação e carrega os dados em conjuntos de N, reduzindo o número de chamadas à base de dados. É uma forma simples de melhorar performance, em particular se o tamanho dos conjuntos for similar ao tamanho das páginas de dados que apresentamos ao utilizador caso em que reduzimos significativamente o número de queries: uma para obter as entidades raiz, e uma por cada colecção que acedemos.

Para tal, usando FluentNHibernate, apenas precisamos de declarar um override para a entidade e indicar que a relação tem um BatchSize do tamanho que pretendemos.

```
public class ArtigoOverride : IAutoMappingOverri-
de<Artigo>
{
public void Override(AutoMapping<Artigo> mapping)
    {
       mapping.HasMany(a => a.Autores)
            .BatchSize(10);
    }
}
```

BatchSize é o número de instâncias de uma relação que são carregadas em simultâneo em vez de separadas. Configurado desta forma, o seguinte código:

```
foreach (var artigo in session.QueryOver<Artigo>
().List())
{
   var autores = artigo.Autores.Select(
                                 a => a.Nome);

    Console.Out.WriteLine(
           "Artigo {0} com autores {1}",
           artigo.Titulo,
           string.Join(", ", autores.ToArray()));
}
```

Passa de :

Short SQL	Row Count
begin transaction with isolation level: Unspecified	
SELECT ... FROM [Artigo] this_	4
SELECT ... FROM [Autor] autores0_ WHERE autores0_.Artigo_id = 2525	0
SELECT ... FROM [Autor] autores0_ WHERE autores0_.Artigo_id = 2526	0
SELECT ... FROM [Autor] autores0_ WHERE autores0_.Artigo_id = 2527	2
SELECT ... FROM [Autor] autores0_ WHERE autores0_.Artigo_id = 2528	0
commit transaction	

Para:

Short SQL	Row Count
begin transaction with isolation level: Unspecified	
SELECT ... FROM [Artigo] this_	4
SELECT ... FROM [Autor] autores0_ WHERE autores0_.Artigo_id in (101,102,103,104)	2
commit transaction	

Como podemos ver, em vez de uma chamada para obter o artigo e uma para cada autor, apenas fazemos uma chamada para obter o artigo, e outra para carregar os autores em batch.

Futures

Existe também o caso em que se pretende obter vários dados que não estão relacionados entre si. Este caso acontece frequentemente quando uma página ou ecrã mostra dados provenientes de várias tabelas ou componentes do software, como um ecrã de resumo, ou uma caixa de informação geral da aplicação.

Se quisermos mostrar numa página um índice de edições com número de artigos em cada uma, um índice com os artigos da edição actual e um artigo em destaque com autores, a solução mais directa passa por fazer três queries separados, resultando em três round-trips á base de dados:

```
var edicoes = session.QueryOver<Edicao>()
    .List();

var edicaoComArtigos = session
    .QueryOver<Edicao>()
    .Fetch(e => e.Artigos).Eager
    .WithSubquery.WhereProperty(e =>
           e.DataEdicao)
    .In(QueryOver.Of<Edicao>()
           .Select(e => e.DataEdicao)
           .OrderBy(e =>
                 e.DataEdicao).Desc.Take(1))
    .SingleOrDefault();

var artigoComAutor = session.QueryOver<Artigo>()
    .Where(a => a.Id == idTerceiroArtigo)
    .Fetch(a => a.Autores).Eager
    .SingleOrDefault();

foreach (var edicao in edicoes)
{
    Console.Out.WriteLine(
           "Edicao {0} ",
           edicao.DataEdicao);
}

var artigos = edicaoComArtigos.Artigos.Select(a
=> a.Titulo).ToArray();

Console.Out.WriteLine(
     "Edicao {0} - Artigos {1}",
      edicaoComArtigos.DataEdicao,
      string.Join(", ", artigos));

var autores = artigoComAutor.Autores.Select(a =>
a.Nome).ToArray();

Console.Out.WriteLine(
     "Artigo {0} com autores {1}",
      artigoComAutor.Titulo,
      string.Join(", ", autores));
```

Usando a ferramenta NHProf para ver os pedidos que são feitos à base de dados, vemos que são feitos 3 pedidos distintos:

SELECT ... FROM [Edicao] this_	2	0 ms / 0 ms
SELECT ... FROM [Edicao] this_ left outer join [Artigo] artigos... WHERE this_.DataEdicao in (Select TOP (1) this_0_...	4 / 4	15 ms / 19 ms
SELECT ... FROM [Artigo] this_ left outer join [Autor] autores2... WHERE this_.Id = 3638	2	14 ms / 15 ms

Na realidade podemos diferir a definição das operações de acesso a dados da sua execução. Podemos ter assim uma fase de preparação, onde se criam os queries que se pretendem fazer, e uma segunda fase onde se mostram os resultados.

Desta forma, podemos preparar todos os queries e de uma só vez fazer todos os pedidos à base de dados.

É este comportamento que os Futures nos permitem. A alteração ao código é mínima. Usamos Futures em vez de List, e *FutureValue* em vez de *SingleOrDefault*:

```
var edicoes = session.QueryOver<Edicao>()
    .Future();

var edicaoComArtigos = session
    .QueryOver<Edicao>()
    .Fetch(e => e.Artigos).Eager
    .WithSubquery.WhereProperty(e =>
           e.DataEdicao)
    .In(QueryOver.Of<Edicao>()
           .Select(e => e.DataEdicao)
           .OrderBy(e => e.DataEdicao)
                 .Desc.Take(1))
    .FutureValue();

var artigoComAutor = session.QueryOver<Artigo>()
    .Where(a => a.Id == idTerceiroArtigo)
    .Fetch(a => a.Autores).Eager
    .FutureValue<Artigo>();

foreach (var edicao in edicoes)
{
    Console.Out.WriteLine(
           "Edicao {0} ",
           edicao.DataEdicao);
}

var artigos = edicaoComArtigos.Value
                 .Artigos.Select(a => a.Titulo)
                 .ToArray();

Console.Out.WriteLine(
           "Edicao {0} - Artigos {1}",
           edicaoComArtigos.Value.DataEdicao,
           string.Join(", ", artigos));

var autores = artigoComAutor.Value.Autores.Select
(a => a.Nome).ToArray();

Console.Out.WriteLine(
           "Artigo {0} com autores {1}",
           artigoComAutor.Value.Titulo,
           string.Join(", ", autores));
```

“Desta forma, podemos preparar todos os *queries* e de uma só vez fazer todos os pedidos à base de dados.”

Enquanto os queries anteriores retornavam logo uma lista, o resultado de um Future ou de um FutureValue é apenas a promessa de que no futuro quando precisarmos dos dados eles estarão lá.

NHibernate - Técnicas para software melhor e mais rápido

O que o NHibernate faz é registar internamente todos os Futures pedidos. Assim que um é concretizado (quer por iterar sobre um IEnumerable de um Future, quer por aceder ao valor de um FutureValue), são enviados todos os queries registados para a base de dados e obtidos os resultados de uma só vez.

O sql gerado é exactamente o mesmo, mas foi todo enviado na mesma chamada, minimizando assim os problemas de latência no acesso aos dados.

Vemos então que em vez de três chamadas distintas fazemos apenas uma chamada que retorna três conjuntos de resultados diferentes:

SELECT ... FROM [Edicao] this_	2 / 4 / 2	4 ms / 5 ms
SELECT ... FROM [Edicao] this_ left outer join [Artigo] artigos... WHERE this_.DataEdicao in (Select TOP (1) this_0_...		
SELECT ... FROM [Artigo] this_ left outer join [Autor] autores2... WHERE this_.Id = 3638		

De reparar que por omissão o Future assume que se retorna uma lista de entidades sobre as quais estamos a fazer o *QueryOver*, e o *FutureValue* assume que se retorna uma entidade. Como vamos ver mais à frente com projecções, também é possível obter instâncias que não as indicadas inicialmente como argumento de tipo ao *QueryOver*.

Esta funcionalidade tem a grande vantagem de poder reduzir grandemente o número de chamadas à base de dados com alterações mínimas ao código.

Outro caso de uso prende-se com o carregamento de várias relações da mesma entidade.

Assumindo que a classe **Edicao** passa a ser a seguinte:

```
public class Edicao
{
  public virtual DateTime DataEdicao {get;set;}
  public virtual Artigo TemaDeCapa {get;set; }
  public virtual IList<Artigo> Artigos{get;set;}
  public virtual IList<Patrocinador>
                    Patrocinadores { get; set; }
}


public class Patrocinador
{
    public virtual int Id { get; set; }
    public virtual string Nome { get; set; }
}
```

Se nós pretendemos mostrar uma edição com todos os artigos e todos os patrocinadores, temos várias hipóteses:

- Aproveitamos o lazy-loading e fazemos vários pedidos à base de dados:
- Fazemos o eager-loading de todas as relações, resultando num query com um número elevado de joins:
- Usamos Futures para carregar em queries diferentes relações diferentes da mesma entidade.

A terceira opção acaba por ser a que obtém tipicamente melhores resultados e é concretizada da seguinte forma:

```
var edicao = session.QueryOver<Edicao>()
    .Where(e => e.DataEdicao == dataEdicao)
    .Fetch(e => e.Artigos).Eager.FutureValue();

session.QueryOver<Edicao>()
    .Where(e => e.DataEdicao == dataEdicao)
    .Fetch(e => e.Patrocinadores)
       .Eager.FutureValue();

var artigos = edicao.Value
      .Artigos.Select(a => a.Titulo).ToArray();

Console.Out.WriteLine(
      "Edicao {0} - Artigos {1}",
      edicao.Value.DataEdicao,
      string.Join(", ", artigos));

var patrocinadores = edicao.Value
      .Patrocinadores.Select(a => a.Nome)
      .ToArray();

Console.Out.WriteLine(
      "Artigo {0} com autores {1}",
      edicao.Value.DataEdicao,
      string.Join(", ", patrocinadores));
```

Saliente-se que usamos apenas o valor do primeiro QueryOver. O segundo é apenas usado para carregar os dados da colecção de forma indirecta.

Isto é possível uma vez que o NHibernate implementa um *identity map* dentro da sessão. Quando acedemos ao valor do primeiro Future despoletamos todos os Futures registados até ao momento na sessão. Ao concretizar os objectos resultantes do segundo query, o NHibernate encontra a entidade já na sessão e preenche a colecção que foi carregada com os dados desse query, que incluem a relação Patrocinadores.

Desta forma conseguimos popular as relações necessárias apenas com uma chamada à base de dados e um conjunto pequeno de queries com apenas um join cada um:

Short SQL	Row Count	Duration
begin transaction with isolation level: Unspecified		
SELECT ... FROM [Edicao] this_ left outer join [Artigo] artigos... WHERE this_.DataEdicao = '2011-07-01T00:00:00.00'	5 / 2	2 ms / 4 ms
SELECT ... FROM [Edicao] this_ left outer join [Patrocinador] p... WHERE this_.DataEdicao = '2011-07-01T00:00:00.00'		
commit transaction		

Como reduzir o tamanho de cada chamada

Até agora trabalhamos com as entidades completas. No entanto existem muitas operações ou casos de uso que necessitam apenas de parte das entidades ou de dados agregados. Ir buscar todos os artigos de uma edição quando apenas pretendemos uma contagem de artigos da edição é improdutivo quando podemos fazer esses cálculos directamente na base de dados e obter apenas os dados que precisamos. Para este efeito usamos projecções em conjunto com transformadores.

Count, Max, Aggregate

Os casos mais simples são aqueles em que pretendemos obter um valor resultante de uma query:

```
var edicaoMaisAntiga = session.QueryOver<Edicao>
().Select(Projections
      .Min<Edicao>(e => e.DataEdicao))
      .SingleOrDefault<DateTime>();
```

Neste caso estamos a procurar o valor mínimo de todas as datas de edição. Como este query devolve um valor do tipo DateTime, precisamos de indicar isso através do argumento de tipo do método SingleOrDefault.

É claro que podemos usar um Future para os casos em que pretendemos obter vários valores sem ter a penalização de várias chamadas à base de dados:

```
var edicaoMaisRecente = session.QueryOver<Edicao>
().Select(Projections
      .Max<Edicao>(e => e.DataEdicao))
      .FutureValue<DateTime>();
```

Outra situação onde projecções são úteis é em subqueries, onde podemos projectar um valor para usar numa condição:

```
Artigo artigo = null;

var numeroArtigosEdicaoMaisRecente = session
     .QueryOver<Edicao>()
     .WithSubquery
     .WhereProperty(e => e.DataEdicao)
     .In(QueryOver.Of<Edicao>()
        .Select(e => e.DataEdicao)
        .OrderBy(e => e.DataEdicao).Desc.Take(1))
    .JoinQueryOver(e => e.Artigos, () => artigo)
    .SelectList(q =>
           q.SelectCount(() => artigo.Id))
    .FutureValue<int>();
```

Neste caso estamos a seleccionar o identificador da edição mais recente, e obter o numero de artigos dessa edição.

Viewmodels

O segundo caso de uso que exemplifico para projecções prende-se com situações onde temos uma ou varias entidades com um número grande de campos e onde necessitamos apenas de parte dos campos, ou de dados agregados em conjunto com alguns campos.

Da mesma forma que projectamos um valor de um resultado de um query podemos também projectar uma lista de valores:

```
IList<Artigo> artigos = null;

var edicoes = session.QueryOver<Edicao>()
      .JoinQueryOver(
           e => e.Artigos,
           () => artigos,
           JoinType.LeftOuterJoin)
      .SelectList(
           q => q.SelectGroup(e =>e.DataEdicao)
      .SelectCount(e => e.Artigos)
    )
    .List<object[]>();

foreach (var edicao in edicoes)
{
    Console.Out.WriteLine(
           "Edicao {0} com {1} artigos",
           edicao[0],
           edicao[1]);
}
```

Neste caso estamos a projectar a data de edição e o número de artigos que aparecem para cada edição. O sql gerado não tem surpresas:

```
SELECT   this_.DataEdicao        as y0_,
         count(this_.DataEdicao) as y1_
FROM     [Edicao] this_
         inner join [Artigo] artigos1_
           on this_.DataEdicao = arti-
gos1_.Edicao_id
GROUP BY this_.DataEdicao
```

No entanto o acesso aos dados projectados torna-se propício a erros, visto retornar uma lista de vectores de objectos.

Seria muito mais fácil se pudéssemos usar uma classe intermédia que armazenasse os campos obtidos. E tal é possível recorrendo a aliases e a um ResultTransformer.

Com a seguinte classe:

```
public class EdicaoComNumeroArtigos
{
  public virtual DateTime DataEdicao {get;set;}
  public virtual int NumeroArtigos {get;set;}
}
```

Podemos alterar o query anterior para associar um alias a cada um dos campos e transformar os resultados em instâncias da classe EdicaoComNumeroArtigos:

```
IList<Artigo> artigos = null;
EdicaoComNumeroArtigos viewModel = null;

var edicoes = session.QueryOver<Edicao>()
    .JoinQueryOver(
            e => e.Artigos,
            () => artigos,
            JoinType.LeftOuterJoin)
    .SelectList(q => q.SelectGroup(
            e => e.DataEdicao)
            .WithAlias(() => viewModel.DataEdicao)
            .SelectCount(e => e.Artigos).WithAlias
            (() => viewModel.NumeroArtigos)
    )
    .TransformUsing
(Transformers.AliasToBean<EdicaoComNumeroArtigos>
())
    .List<EdicaoComNumeroArtigos>();

foreach (var edicao in edicoes)
{
    Console.Out.WriteLine(
            "Edicao {0} com {1} artigos",
            edicao.DataEdicao,
            edicao.NumeroArtigos);
}
```

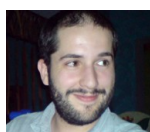
Utilizamos expressões para indicar os nomes dos campos, sendo que na realidade o que o NHibernate faz é apenas associar nomes dos campos à projecção. Depois o transformador AliasToBean é que tem o trabalho de para cada campo retornado do query procurar uma propriedade com um setter público e colocar o valor do campo nessa propriedade. Isto quer dizer que é possível indicar nomes inválidos ou ter outros problemas semelhantes nos queries, no entanto são erros facilmente detectados com testes de integração que correm o query, visto o NHibernate identificar logo as irregularidades.

Conclusão

Neste artigo analisamos algumas opções e funcionalidades que nos permitem melhorar as características de performance das nossas aplicações, quer por um carregamento mais inteligente dos dados, como pela redução do tamanho de dados que é transferido da base de dados para a aplicação.

Existem ainda algumas funcionalidades que não foram mencionadas, como interceptores, filtros, cache de segundo nível e concorrência optimista.

Ver artigo Revista PROGRAMAR - edição 30 de Agosto 2011: **NHibernate - do Import Package à primeira iteração**

AUTOR

Bruno Lopes fez o curso de Engenharia Informática no IST, e neste momento conta com mais de 5 anos de experiência profissional em IT, em particular nas áreas de desenvolvimento de software web-based. Actualmente é co-fundador da weListen Business Solutions, trabalhando no produto InnovationCast, e participa activamente na comunidade NetPonto, apresentando temas como NHibernate, RavenDB ou IoC.

Tem blog em http://blog.brunomlopes.com, twitter em @brunomlopes e linkedin em http://pt.linkedin.com/in/brunomlopes

Elege o melhor artigo desta edição

Revista PROGRAMAR

http://tiny.cc/ProgramarED31_V

A PROGRAMAR

Geração de números aleatórios (Parte 1)

Diferenças ao Desenvolver em Windows Azure

Programação Orientada a Objectos em Java 6

DataBinding em Silverlight 4

GNA - GERAÇÃO DE NÚMEROS ALEATÓRIOS (Parte 1)

A partir deste artigo, serão apresentadas em quatro partes, informações sobre a geração de números aleatórios ou *random number generator* em computadores.

São muitos os métodos existentes para a geração de números aleatórios. Podendo-se destacar: o método do meio do quadrado (tema deste artigo), o método produto do meio, o randu e o método linear congruente.

INTRODUÇÃO

As discussões sobre geração de números aleatórios originam-se a partir das ideias dos filósofos gregos Demókritos e Epikurus, indicado por Silveira (2001) ao mostrar as propostas de:

aleatoriedade objetiva, proposta por Epikurus, que afirma existir na Natureza o aleatório verdadeiro, o qual ocorre a partir do desconhecimento das causas;

aleatoriedade subjetiva, proposta por Demókritos, que afirma ser a Natureza determinista, onde tudo ocorre devido a uma causa.

Para Doricio (1998) "não existe o que se possa chamar de número aleatório", o que corrobora com a afirmação indicada por Schildt (1990), ao dizer que "o termo geração de números aleatórios é um absurdo".

Apesar da concordância, ou não, sobre este tema, os computadores eletrónicos de alguma forma produzem sequências de valores numéricos, chamados "números aleatórios", a partir de duas abordagens:

- geração de números pseudo-aleatórios (PRNG – *Pseudo-Random Number Genarator*);
- geração de números verdadeiramente aleatórios (TRNG – *True Ramdom Number Generator*).

O mecanismo PRNG é a forma mais comum encontrada nos computadores eletrónicos. A palavra "pseudo" sugere, neste contexto, o uso de valores numéricos que não são gerados de forma aleatória, uma vez que são obtidos por meios artificiais a partir de algoritmos matemáticos elaborados para esta finalidade.

O mecanismo TRNG não é comum ou de fácil acesso (os equipamentos para esta finalidade são muito caros para utilizadores menos científicos), uma vez que os valores chamados aleatórios são obtidos a partir de fenómenos oriundos da natureza, como: ruídos ambientais, alterações climáticas, entre outros fenómenos. A geração de números aleatórios, sejam PRNG ou TRNG, é são necessárias em vários tipos de aplicação, destacando-se algumas, como as apontadas por Doricio (1998) e Zenil (2011):

- simulação – aplicação de modelos científicos na simulação de fenómenos naturais;
- amostragem – aplicação de modelos científicos baseados numa parte menor de um todo maior, que será analisada;
- análise numérica – aplicação de modelos determinísticos numéricos para a economia de tempo de computação;
- programação de computadores – aplicação de valores aleatórios para testes de eficiência algorítmica computacional;
- teoria de decisão – auxiliar a tomada de decisão com base na escolha aleatória de dados;
- recreação – aplicação em jogos de azar.

A geração de números pseudo-aleatórios depende da escolha adequada dos parâmetros usados para a geração do número em si. Um desses parâmetros, que merece atenção, é o chamado "semente" (DIAS, 2005), o qual é caracterizado por ser o valor inicial a ser usado para a geração dos demais valores "aleatórios" subsequentes por meio de algum algoritmo específico para esta finalidade. Segundo Dias (2005) a escolha da semente deve ser realizada com muito cuidado, pois "a utilização de uma mesma semente […] entre diferentes aplicações […] pode ocasionar erros bastante comuns que levam a conclusões que podem representar de forma incorreta o comportamento dessas aplicações".

Para a geração de números pseudo-aleatórios existe de um grande conjunto de algoritmos. Neste estudo, em particular é apresentado o método do meio do quadrado.

MÉTODO DO MEIO DO QUADRADO

A geração de números pseudo-aleatórios a partir do método do meio do quadrado (*middle square method*) foi propostao em 1946 por John von Neumann, sendo este algoritmo considerado o primeiro método de geração de números "aleatórios"

(GIORDANO, FOX, WEIR, 2009 – p.184; TÖRN, 2001).

No método do meio do quadrado, foi proposto por Von Neumann, o uso de um valor semente que, elevado ao quadrado, gera um resultado. Desse resultado é retirado do

meio um número com o mesmo tamanho em dígitos, do valor usado para a semente. A partir do novo valor repete-se a operação para a obtenção de um novo valor e assim por diante (TÖRN, 2001; WHELAN, 2011; STUBBE, 2011). Apesar de engenhoso, o método apresenta algumas falhas. Segundo Knuth (KNUTH, 1969 – p. 4), várias pessoas fizeram uso do método do meio do quadrado no início da década de 1950 tendo sido utilizadas sementes com tamanho de quatro dígitos, sendo este o tamanho de semente mínimo recomendado. Descobriu-se que alguns valores levam a sequências infinitas de geração de valores pseudo-aleatórios, ao apresentarem valores como: 6100, 2100, 4100 e 8100. A partir daí, o cálculo do meio do quadrado efetua a repetição infinita da mesma sequência. Descobriu-se também, que alguns valores usados levam à obtenção de valor zero como resultado do meio do quadrado, impossibilitando a continuidade do uso do método, tais como os múltiplos de 1000 para as sementes de quadro dígitos. Um bom gerador de números "aleatórios" deve gerar sequencias infinitas, sem resultar em zero. Não deve repetir valores em sequências. No entanto, para aplicações que não requerem grande grau de precisão o método do meio do quadrado pode ser útil.

Tomando-se por como base o valor de semente 5678 (quatro dígitos de tamanho) e elevando-se este valor ao quadrado, obter-se-á como resultado o valor 32239684 (oito dígitos de tamanho). Do quadrado obtido são retirados os quatro dígitos do meio desse quadrado. Assim sendo, do ao valor 32239684 tira-se o valor 2396.

Na sequência, a partir do valor 2396 calcula-se o seu quadrado, que resultará no valor 5740816 (sete dígitos de tamanho). No entanto, este segundo quadrado possui sete dígitos de tamanho e devido a isto necessita ser equacionado para que possua oito dígitos. Assim, basta acrescentar um valor zero à esquerda do quadrado, de forma que fique assim representado 05740816. Deste ajuste, pega -se nos quatro dígitos do meio, que neste caso é o valor 7408 e prossegue-se com a aplicação da técnica de forma sequencial até o ponto desejado ou quando o meio do quadrado for 0000.

ALGORITMO DO MEIO DO QUADRADO

A aplicação do método pode ser efetivada a partir do algoritmo seguinte:

N = entrada do número que representa a semente (> que 4 - máx: 5 p/ 32 bits | 15 p/ 64 bits).

d = tamanho em dígitos do valor N: d = tamanho(N).

t = tamanho em dígitos do quadrado de N: t = tamanho(N^2).

tqd = tamanho em dígitos do quadrado de N, sua obtenção depende de algumas condições.

```
SE ((d é par) .E. (t é impar))
     .OU.
    ((d é impor) .E. (t é impar))
    Acrescentar 00 à esquerda de tqd
  SENÃO
    SE (d é par) .E. (t é par) ENTÃO
      Manter tqd como está
```

tq = tamanho em dígitos do valor gerado para tqd.

```
SE (t é impar) ENTÃO
  tq = d.2+1
SENÃO
```

m = posição inicial para extração do quadrado do meio a partir da quantidade de dígitos tqd.

$$m = \left[\frac{d.2 - \left(\frac{d.2}{2} \right)}{2} \right]$$

ns = valor da próxima semente (valor pseudo-randômico gerado) após extração do quadrado do meio.

$$ns = \left(\begin{matrix} tq \\ m, d \end{matrix} \right)$$

Com base no algoritmo do método do meio do quadrado proposto por von Neumann e das etapas conjugadas , é possível obter os valores indicados na Tabela 1 a partir do fornecimento de uma semente que tenha entre 4 e 7 dígitos de tamanho.

O tamanho máximo em dígitos para a semente deve ser 5 para valores de 32 *bits* ou 15 para valores de 64 *bits*.

(d)	(t)	(tqd)	(tq)	(m)	(ns)
4	7	09999999	8	3	8 3,4
	8	99999999			
5	9	00999999999	11	4	11 4,5
	10	09999999999			
6	11	099999999999	12	4	12 4,6
	12	999999999999			
7	13	009999999999999	15	5	15 5,7
	14	099999999999999			

CONCLUSÃO

Neste artigo foi apresentado o método de geração de números pseudo-aleatórios meio do quadrado, que é considerado por vários especialistas da área da computação como sendo um método não eficiente. No entanto, é preferível ter possuir em mãos um método não eficiente de geração de números aleatórios do que não possuir absolutamente nenhum método.

É claro que para uma aplicação que exija maior precisão, será necessário considerar algum outro método.

No próximo artigo será discutido o método quadrado do meio, que é uma variação do método apresentado nesta parte.

BIBLIOGRAFIA

DIAS, G. N. *A Influência da Semente na Geração de Seqüências de Números Aleatórios através de Geradores de Números (Pseudo) Aleatórios*. Rio de Janeiro: Universidade Federal do Rio de Janeiro. 2005. Disponível em: <http://bit.ly/qznXtW>. Acesso em: 29 jun. 2011, 10:52:35.

DORICIO, J. L. *Número Aleatórios e Apli-cações*. São Paulo: Universidade Federal de São Carlos, Departamento de Matemática. 1998.

GIORDANO, F. R.; FOX, W.F. & WEIR, M. D. *A first course in mathematical modeling*. 4. ed. California Books/Cole. P. 184, 2009.

KNUTH, D. E. *The Art of Computer Pro-gramming: series in computer science and information processing*. 2d ed. Indiana: Addison-Wesley. 1981.

REYS, A. E. L.; MACHADO, A. A.; FERREIRA, D. F.; DEMÉTRI, C. B. & RIBEIRO, P. J. *Sistema Galileu de Educação Estatística*. São Paulo: Universidade de São Paulo, ESALQ. 2011. Disponível em: http://bit.ly/q5jbT8

. Acesso em: 1 jul. 2011, 08:17:25.

SCHILDT, H. *Turbo C Avançado: Guia do Usuário*. Rio de Janeiro: McGraw-Hill, 1990. 475 p.

SILVEIRA, J. F. P. da. *Tipos de Aleatori-edade*. Rio Grande do Sul: Universidade Federal do Rio Grande do Sul, Departamento de Matemática. 2001. Disponível em: <http://bit.ly/r2YYis> . Acesso em: 29 jun. 2011, 11:50:23.

STUBBE, J. *An Introduction to Random Number Generation and Simulation. École Polytechnique Fédérale de Lausanne*.

Disponível em: http://bit.ly/nvkNaP. Acesso em: 1 jul. 2011, 08:59:12.

TÖRN, A. Probabilistic Algorithms: Spring 2001 Course. Åbo Akademi University: Department of Computer Science. 2001.

Disponível em <http://bit.ly/qYxYkQ>. Acesso em: 1 jul. 2011, 08:27:32.

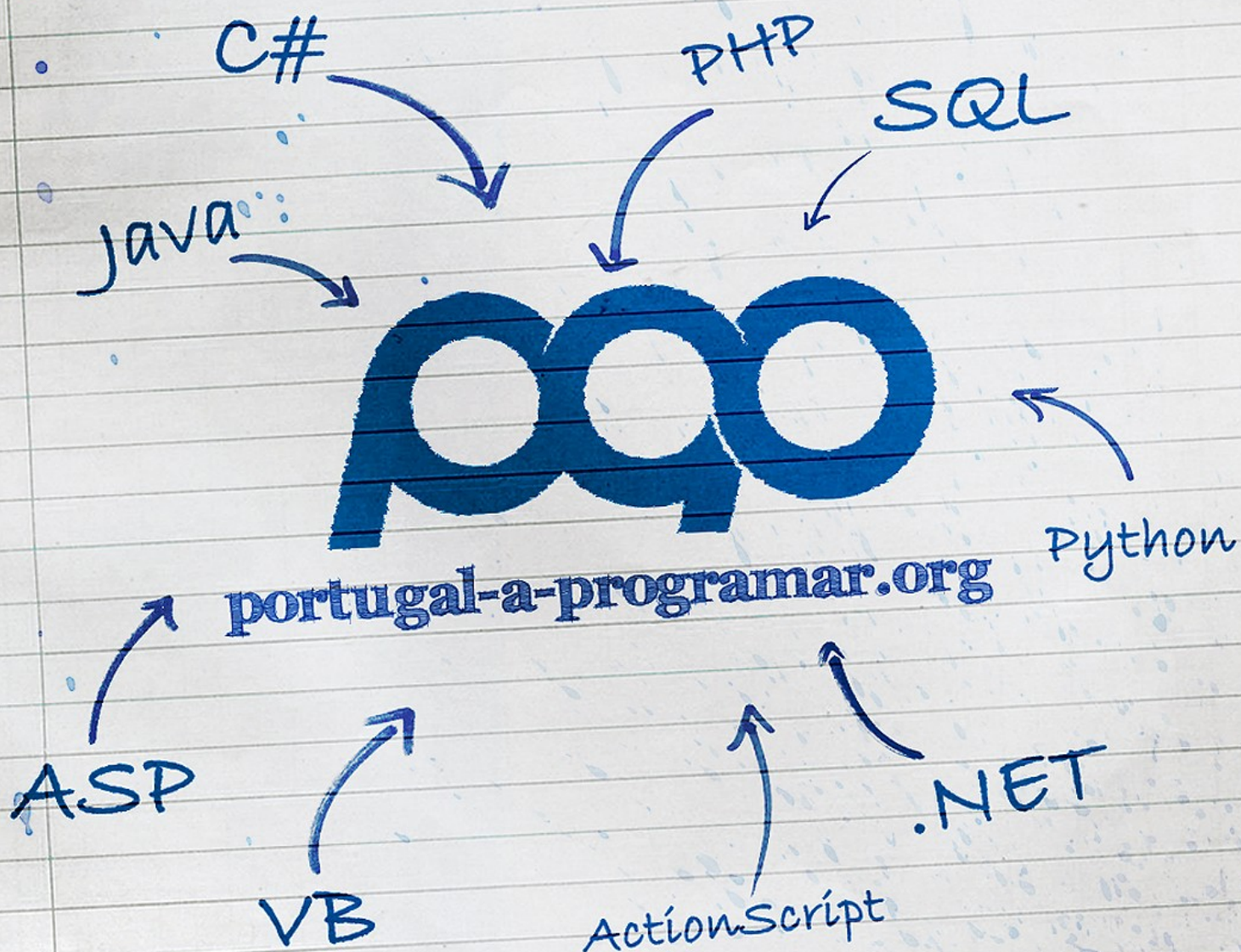
WHELAN, S. F. *Models - Stochastic Models*. University College Dublin, School of Ma-thematical Sciences. Disponível em: http://bit.ly/nIy2nT

Acesso em: 1 jul. 2011, 08:39:54.

ZENIL, H. *John von Neumann's First Pseudorandom Number Generator Pseu-dorandom Number Generator*. Champaign: Wolfram Demonstrations. 2011. Disponível em < http://bit.ly/6C7Q5

JohnVonNeumannsFirstPseudorandomNumberGenerator>. Acesso em: 15 ago. 2011, 08:23:56.

AUTOR

Augusto Manzano, natural da Cidade de São Paulo, tem experiência em ensino e desenvolvimento de programação de software desde 1 986. É professor da rede federal de ensino no Brasil, no Instituto Federal de Educação, Ciência e Tecnologia. É também autor, possuindo na sua carreira várias obras publicadas na área da computação.

Programar para Windows Azure

O Windows Azure é uma plataforma de alta disponibilidade e alta escalabilidade que é fornecida como serviço e tem por isso algumas características diferentes em relação às plataformas alvo tradicionais.

Neste artigo vamos tentar alertar o leitor para alguns detalhes que encontramos em Windows Azure Compute, SQLAzure e Windows Azure Storage bem como alguns cuidados que deveremos ter quando desenvolvemos para estes serviços.

Windows Azure Compute

I/O Performance

Existem cinco tipos de instâncias de Windows Azure Compute que permitem a execução de várias cargas de trabalho e aplicações com vários níveis de complexidade.

Tamanho da	CPU	Memória	Local Storage	Largura de Banda
Extra Small	1.0 GHz	768 MB	20 GB	5 Mbps
Small	1.6 GHz	1.75 GB	225 GB	100 Mbps
Medium	2 x 1.6	3.5 GB	490 GB	200
Large	4 x 1.6 GHz	7 GB	1,000 GB	400 Mbps
Extra large	8 x 1.6 GHz	14 GB	2,040 GB	800 Mbps

Cada instância de Windows Azure Compute representa um servidor virtual. Apesar de muitos dos recursos serem dedicados a essa instância em particular, alguns recursos associados à performance de I/O, tais como largura de banda e disco rígido, são partilhados entre as instâncias presentes na mesma máquina física. Durante os períodos em que os recursos partilhados estão a ser muito utilizados poderemos ver a performance de I/O baixar para os níveis mínimos anunciados mas, em contrapartida, sempre que os recursos partilhados tiverem um nível de utilização inferior veremos essa performance aumentar.

Quanto maior for o tamanho da instância maior será a performance mínima de I/O e consequentemente essa performance será também mais consistente ao longo do tempo.

Capacidade VS Quantidade

Em Windows Azure podemos escalar horizontalmente adicionando mais máquinas virtuais ou escalar verticalmente usando máquinas virtuais de maior capacidade.

Dependendo do tipo de aplicação e da carga de trabalho que cada aplicação, deverá ser escolhida a quantidade e capacidade das instâncias de modo a optimizar os custos e desempenho. A decisão poderá ser tomada tendo por base algumas das seguintes características:

Maior capacidade

- Comunicação mais rápida e mais barata entre processos (mesma máquina)
- Pode sair mais barato nos casos em que se usam muito Queues porque podemos fazer Queue.GetMessages(32) e processar em paralelo
- Máquinas com maior capacidade têm uma maior e mais consistente performance de I/O

Maior quantidade

- Permite uma escalabilidade mais granular e consequentemente mais barata
- Maior tolerância a falhas. A falha de uma máquina virtual têm menor impacto
- Temos que usar Azure Storage para comunicar entre processos
- Local Storage

O Fabric Controller pode a qualquer momento destruir a máquina virtual e substituir por uma nova. Isso pode acontecer, por exemplo, sempre que seja detectado falhas na aplicação, falhas no hardware, ou simplesmente porque precisa de realizar actualizações ao sistema operativo.

Por essa razão a informação armazenada em local storage é considerada volátil. Esta situação deverá ser levada em conta, principalmente nos casos em que existe migração de aplicações para o Windows Azure. Algumas aplicações desenhadas para correr apenas num servidor usam local storage para guardar alguns ficheiros de forma persistente.

Como já vimos, em Windows Azure essa informação é considerada volátil. Em contrapartida a migração de aplicações desenhadas para executar em web farms normalmente não apresentam este problema.

Outro alerta em relação ao uso de *local storage* tem a ver com a visibilidade da informação. Tudo o que for escrito para *local storage* só vai ser visto pela instância actual.

Estamos a desenvolver para um ambiente altamente escalável e na maioria dos casos manter a informação privada não é a melhor opção. Existem casos em que é vantajoso fazer cache local de alguns dados mas na maioria dos cenários deve ser usado um tipo de storage persistente e partilhado.

Roles

As instâncias são pagas por hora de relógio

As instâncias são facturadas com base nas horas de relógio em que estiveram implementadas. As horas parciais serão facturadas como horas completas. Após esta frase podermos ficar a pensar que se for usada uma instância durante 30 minutos será facturada 1 hora, mas este pensamento pode não ser verdade.

Se fizer *deployment* de uma instância às 16h50 e parar essa instância às 17H10 irá pagar duas horas. Uma hora pela utilização entre as 16h50 e as 17H00 e outra hora pela utilização entre as 17h00 e as 17h10.

Instâncias que estejam implementadas menos de 5 minutos dentro de uma hora não serão contabilizadas.

Cada novo deployment conta uma nova hora

Deveremos evitar implementar, apagar e voltar a implementar instâncias sempre que possível. Cada nova implementação adiciona uma hora por cada instância à conta.

Imagine que implementou uma instância às 15h10 e parou a mesma às 15h20. Se fizer nova implementação o App Fabric irá arrancar uma nova máquina e será cobrada uma nova hora completa.

Windows Azure Database

Limitações

O SQLAzure foi desenhado tendo como base o SQLServer sendo por isso natural que partilhe muitas das suas funcionalidades. O SQLAzure suporta múltiplas bases de dados bem como a maioria dos objectos presentes na versão *on-premisses* tais como tabelas, vistas, procedimentos armazenados, funções, restrições (*constraints*) e *triggers.*

No entanto existem algumas diferenças entre as quais se destacam as seguintes:

Não existe Backup e Restore

Uma das funcionalidades que ainda não existe em SQLAzure é a capacidade de realizar *backup* ou *restore* de uma base de dados. Apesar do estarmos na presença de um serviço de alta disponibilidade onde "não existe" a preocupação com a perca de informação, continuamos a ter que lidar com possíveis corrupções nas bases de dados devido a erros nas aplicações.

Não existe cross-database querying

Em SQL Azure nós temos a capacidade de alugar bases de dados e não servidores de bases de dados. Não existe garantia que todas as bases de dados que alugamos estão no mesmo servidor. A ideia principal que deveremos reter neste caso é que não existe conectividade entre bases de dados. Se for necessário combinar resultados de mais do que uma base de dados teremos que resolver essa questão na aplicação. Deveremos realizar as consultas a todas as bases de dados e depois, na aplicação, agregar os resultados.

Analysis Services, Replication e Service Broker não estão disponíveis como serviço

Apesar de estes serviços ainda não estarem disponíveis é possível usar SQL Azure como fonte de dados para a versão *on-premisses* de Analysis Services e Integration Services.

Não suporta Database Mirroring ou Failover Clustering

Devido às características de alta disponibilidade do serviço SQL Azure os serviços de suporte a alta disponibilidade da versão *on-premisses* tais como *database mirroring* e *failover cluster* não são necessários e não estão disponíveis.

Clustered Indexes são obrigatórios

Todas as tabelas em SQL Azure necessitam de um ter um *clustered index*. Um *clustered index* é um tipo de índice especial que ordena a forma como os registos são gravados fisicamente. Esta é uma das primeiras preocupações na migração de bases de dados para SQLAzure.

Não existe integração com CLR

SQLAzure não suporta CLR. Qualquer base de dados construída usando CLR não será possível mover para o SQLAzure sem que se proceda a modificações.

Não existe acesso às System Tables

Devido a inexistência de acesso ao hardware que suporta a base de dados em SQLAzure não existe acesso às *system tables* do SQLServer. Também não existe acesso às vistas e procedimentos armazenados do sistema.

SQL Azure necessita de SQL Server Management Studio 2008 R2

Para poder aceder a bases de dados através do SQLServer Management Studio tem que actualizar para a versão 2008 R2. Apesar de as versões anteriores estabelecerem ligação com o servidor SQL Azure o Object Browser não irá funcionar. Podemos usar a versão gratuita desta ferramenta (SQL Server Management Studio Express 2008 R2).

Nem todos os dados são relacionais

Esta afirmação não trás novidade nenhuma mas ganha especial importância no conceito de Cloud Computing e Windows Azure. O Windows Azure dispõe de diferentes suportes ao armazenamento de dados, cada um com as suas características e custos. Ao afirmar que nem todos os dados são relacionais pretendemos alertar para o facto de que deverá ser escolhido o tipo de *storage* certo para o tipo de dados certo.

O serviço SQLAzure é, relativamente a Azure Storage, um serviço mais caro. SQLAzure custa cerca de 10$/GB (7.5€/GB) e TableStorage custa cerca de 0.15$/GB (0,11€/GB). Neste sentido, poderemos como exemplo recomendar o armazenamento de imagens em Blob Storage ou a passagem de dados históricos ou dados de auditoria (Logs) da aplicação para Table Storage.

Procedimentos Armazenados

A execução de procedimentos armazenados não tem custos adicionais. Podemos tirar partido desta situação e usar algum poder de computação do servidor de SQL com o intuito de poupar no número de roles necessários para a execução da aplicação. De notar que não deveremos abusar destes recursos.

Windows Azure Storage

Relativamente a Windows Azure Storage vamos tentar perceber alguns dos custos associados a este serviço e tentar perceber alguns dos cuidados que deverão ser tidos em conta na hora de desenvolver sobre estes serviços.

Em relação a custos com Windows Azure Storage, estes são medidos usando as seguintes variáveis:

- Tráfego: quantidade de dados transferidos de e para a o serviço
- Transacções: quantidade de pedidos feitos ao serviço
- Capacidade: Quantidade de informação armazenada

Tráfego

O tráfego consumido internamente não tem custo associado, ou seja, os dados transferidos entre serviços presentes no mesmo datacenter não contam tráfego. Tráfego que tenha origem ou destino diferente do datacenter onde o storage service foi criado conta tráfego, seja ele externo ou até mesmo outro datacenter Azure.

Transacções

Cada chamada a storage service conta uma transacção e cada transacção tem um custo. Apesar de o custo de cada transacção individualmente ser muito baixo (0.01$/10.000 transacções) deveremos ter esta situação em atenção no desenho das aplicações porque rapidamente se consegue chegar a cenários em que o número de transacções é muito elevado. Alerta-se ainda para o facto de que cada pedido ao storage service conta uma transacção independentemente da origem. Isto significa que os pedidos de serviços presentes no mesmo datacenter também contam transacções.

A maioria dos pedidos ao storage service conta apenas uma transacção mas existem algumas chamadas a métodos das Storage Client Library que podem resultar em várias transacções (vários pedidos):

Upload para Blob Storage

Sempre que existe um pedido de upload para Blob Storage de um ficheiro com um tamanho superior a 32Mb a Storage Cliente Library divide automaticamente o pedido em múltiplos blocos de 4Mb. A cada bloco de 4Mb irá corresponder um pedido PutBlock que contará uma transacção cada. Existe ainda um PutBlockList no final que contará também uma transacção. O tamanho do bloco poderá ser modificado através da propriedade CloudBlobClient.WriteBlockSizeInBytes.

Consultas a tabelas

Sempre que usar a classe CloudTableQuery para realizar consultas a um Azure Table Service terá automaticamente a resolução de *continuations tokens* de modo a obter todos os resultados pretendidos.

Isto significa que sempre que a *query* retornar um *continuation token* irá ser lançado automaticamente um pedido dos restantes resultados e como já vimos anteriormente, cada pedido conta uma transacção.

Table.SaveChanges

Sempre que for realizada uma operação de Add, Update ou Delete sobre um objecto de uma tabela, esse objecto será adicionado ao *datacontext* de modo a que, mais à frente, esse pedido possa ser enviado ao Table Service. Os pedidos não são enviados imediatemente, são enviados quando for chamado o método SaveChangesWithRetries.

Quando esse método é chamado as alterações pendentes são chamadas uma a uma sobre o table service resultando numa transacção cada.

Existe no entanto uma excepção a esta situação. Sempre que as alterações pendentes incidirem numa única PartitionKey de uma tabela poderemos solicitar que as alterações sejam efectuadas utilizando apenas um pedido através da chamada SaveChangesWithRetries (SaveChangesOptions.Batch).

Além de apenas contar uma transacções, submeter as alterações pendentes em batch demora menos tempo e garante processamento transaccional (ou todas são aplicadas ou nenhuma é aplicada).

Queues

Cada elemento numa *Azure Queue* custa 3 transacções

Nos casos mais comuns usar uma *queue* para enviar uma mensagem de um role para outro custa cerca de 3 transacções porque são necessárias as operações de Put, Get e Delete para processar uma mensagem.

Poderemos diminuir o custo cada elemento na *queue* utilizando a operação GetMessages da Queue Service API para obter e processar até 32 mensagens de uma só vez e assim diminuir o custo até 2,03 transacções por elemento na *queue*.

Usar backoff exponencial para tempos de espera

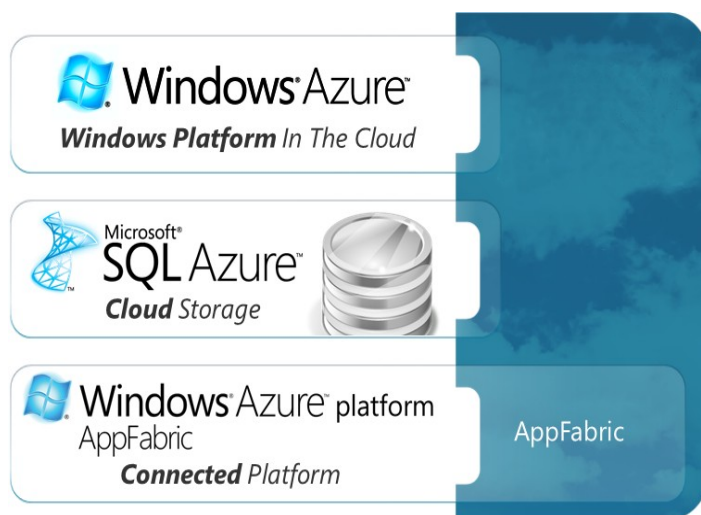
Devido à ausência de notificações nas *queues* é necessário estar constantemente a fazer *pooling* na *queue* para detectar a entrada de novas mensagens.

Imagine que existe um *worker role* com duas instâncias (de modo cumprir o SLA) a fazer *pooling* com um intervalo de 1 segundo numa *queue*. Ao final de um mês haverá um custo de mais de 5$ só derivado ao *pooling*.

Poderá ser implementado um esquema de espera exponencial para reduzir o número de chamadas de *pooling* à *queue*. O incremento de tempo a esperar por cada chamada a uma *queue* vazia e o número máximo de tempo que esse *backoff* poderá atingir deverá ser adaptado tendo em consideração o cenário para qual a *queue* está a ser utilizada.

Conclusão

Através deste artigo esperamos ter conseguido desafiar o leitor para a necessidade de conhecer em detalhe a arquitectura dos serviços e os custos associados de modo a que possa desenhar e desenvolver aplicações para Windows Azure de forma eficiente e eficaz.

Como dica final, no *site* de *billing* podemos ver o estado actual da subscrição até ao final do dia de ontem e desta forma ir verificando os custos das nossas aplicações de modo a não existirem surpresas no final do mês.

AUTOR

Vítor Tomaz é consultor independente na área das tecnologias de informação. Tem especial interesse por Cloud Computing, programação concorrente e segurança informática. É membro de algumas comunidades tais como Portugal-a-Programar, NetPonto, AzurePT, HTML5PT e GASP.

Programação Orientada a Objectos em Java 6

O objectivo deste artigo não é ensinar os conceitos de POO, nem a programar em Java. Assim a leitura do artigo pressupõe que o leitor já saiba os conceitos de POO, e conheça pelo menos a linguagem Java, ou outra de sintaxe semelhante para perceber os exemplos. O objectivo deste artigo é referir as melhores práticas e métodos de programar POO em Java 6.

Parametrização das Colecções

Antes do Java 5, a maneira de implementação de colecções, por exemplo a classe ArrayList poderia levar inicialmente um int e posteriormente uma String. Assim era fácil que fosse gerado um erro em runtime, quando estivéssemos por exemplo, a percorrer a colecção e somar todos os elementos. No entanto agora é possível parametrizar a colecção e dizer para ela só aceitar uma determinada estrutura de dados (ou estruturas descendentes que herdem naturalmente as mesmas propriedades). Assim quando antes poderíamos ter no código algo como:

```
public int exemplo()
{
      ArrayList v = new ArrayList();
      int soma = 0;
      v.add(new Integer(2));
      v.add("Nome X");
      //...
      Iterator x = v.iterator();
      while(x.hasNext())
      soma += Integer.parseInt(x.next().toString
());
      return soma;
}
```

Este código não dá nenhum erro de sintaxe, no entanto qualquer programador que perceba o mínimo de Java verá que ao executar a aplicação dará um erro no ciclo. Porque tentará somar uma sequência de caracteres com um número inteiro. Contudo se aplicarmos um código semelhante no Java 6, graças às parametrizações obteremos um erro de sintaxe.

```
  public int exemplo(){
        ArrayList<Integer> v = new Array-
List<Integer>();
        int soma = 0;
        v.add(new Integer(2));
        v.add("Nome X"); //Aqui dará erro de sin-
taxe
```

O ArrayList é assim genericamente ArrayList<E>, sendo que o E pode ser substituído por qualquer classe, mas não por dados de tipos primitivos (como o int, o double…), mas isso é resolvido de maneira simples, como é explicado mais à frente na secção AutoBoxing.

Podemos aperceber-nos que a grande vantagem das parametrizações, é garantir mais segurança ao programador, de modo a que ele saiba que tipos de coisas estão em vários locais. O tipo Object dá uma sensação de liberdade, porque podemos meter tudo lá dentro, algo como o (void *) do C. Contudo deixa uma responsabilidade acrescida ao programador que terá que ter em atenção sempre o que está realmente lá dentro para não realizar operações não permitidas. Pelo contrário com as parametrizações é possível saber exactamente que operações são permitidas sobre a estrutura de dados, porque ao escrever o código sabemos o que lá estará.

Outra grande vantagem é a inexistência de conversões. No primeiro exemplo, depois de obter o valor da lista era necessário converter para inteiro (e o valor obtido poderia ser inconvertível para inteiro) e só depois somar. No segundo exemplo, o valor devolvido já é do tipo que pretendemos (neste caso um inteiro, o que nós estávamos à espera).

AutoBoxing

Infelizmente os tipos parametrizados não permitem trabalhar com tipos primitivos, algo que é extremamente importante. Aliás muitos dos problemas requerem vectores de inteiros. Será que então a única solução é trabalhar com os Arrays originais? Não. Todos os tipos primitivos possuem a correspondente classe. Por exemplo a classe Integer, corresponde ao tipo primitivo int.

Assim podemos facilmente fazer:

```
ArrayList<Integer> v = new ArrayList<Integer>();
```

A seguir para adicionar elementos teríamos que colocar algo deste género:

```
v.add(new Integer(3));
v.add(new Integer(9));
int r = v.get(1).intValue();
```

O construtor é necessário, uma vez que os tipos não são compatíveis. Aliás o seguinte exemplo, compilado antes do Java 5 não funcionaria correctamente:

```
int i;
Integer j;
i = 1;
j = 2;
i = j;
j = i;
```

Isto porque int é um tipo primitivo e Integer um tipo complexo. Logo não é possível fazer uma atribuição de um tipo primitivo para um tipo complexo. Tanto mais que o que está realmente armazenado em j é um “apontador” para uma instância da classe Integer. Felizmente devido ao AutoBoxing o exemplo a partir de Java 6 funciona perfeitamente, o que nos permite fazer também no ArrayList anteriormente criado algo mais simples:

```
v.add(3);
v.add(9);
int r = v.get(1);
```

Pode parecer uma pequena diferença, mas numa linguagem como Java, onde é muitas vezes necessário escrever grandes linhas de código, isto pode poupar bastante esforço num projecto.

Interfaces

Apesar de já existir antes de Java 5, as interfaces são uma maneira simples de Java ultrapassar as limitações de herança simples, ao invés da múltipla do C++, e ao mesmo tempo evitando grande parte do problema do Diamante da Morte (Diamond of Death em Inglês), onde a múltipla herança pode criar problemas ao compilador e ao programador sobre qual o método a ser executado.

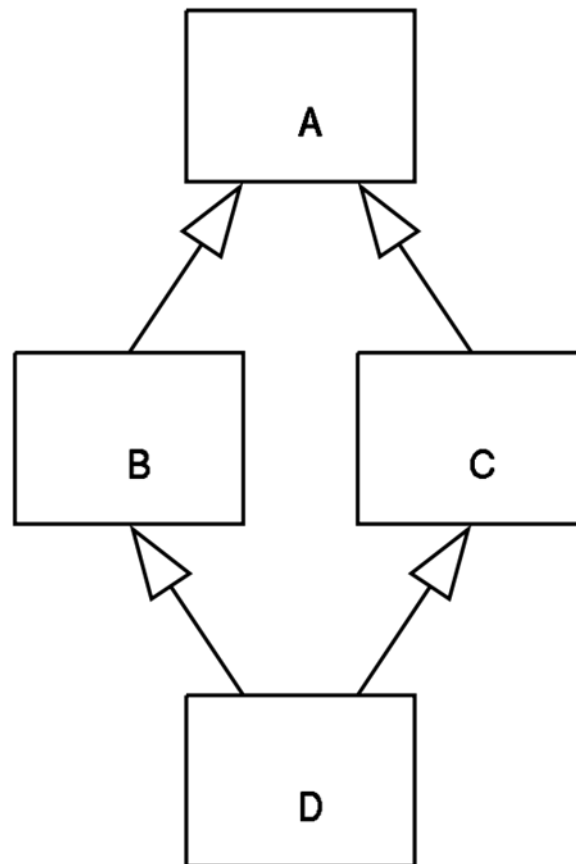



Figura 1: Exemplo do Diamon of Death

Na fig. 1 podemos ver um exemplo em que ambas as classes B e C herdam da classe D. Por sua vez, a classe A herda da B e C simultaneamente. Supondo que a classe D possui o método xpto(), e tanto a classe B como a C na sua implementação reescrevem o método. Supondo que ele é chamado no contexto da classe A, e assumindo que ele não é reescrito na classe A, então qual seria o método a ser chamado? O da classe B ou C? Nesse caso temos o famoso Diamante da Morte… Como não é possível definir comportamentos nas interfaces, apesar de uma classe poder implementar várias interfaces, mesmo que ambas possuam métodos com a mesma assinatura, ele têm que ser definido na classe e é esse que será usado. Apesar de prevalecer o problema de métodos com a mesma assinatura, mas com funções distintas, o que impossibilitava a implementação de ambas as interfaces.

Aqui está um exemplo da declaração de uma interface.

```
public interface Expiravel {
	public bool estaExpirado();
	public GregorianCalendar expiraEm();
}
```

Qualquer classe que implemente esta interface, terá que obrigatoriamente possuir aqueles dois métodos especificados. Assim poder-se-á facilmente percorrer, por exemplo uma lista de objectos de diferentes classes, mas com uma interface em comum. Como é óbvio, os únicos métodos que poderão ser chamados, sem serem necessárias conversões, serão os definidos na interface.

Classes Abstractas

As classes abstractas são muito úteis para possibilitar a poupança de código, e garantir o funcionamento correcto da classe, que apesar de tudo estará incompleta.

Imaginemos por exemplo o caso de uma Biblioteca. Poderia existir uma classe abstracta, para as publicações, que teria informações gerais, como o código de identificação, se estava requisitado, se era possível de requisitar, quantidade, estado, entre outros atributos. No entanto possuiria métodos abstractos, que têm que ser codificados obrigatoriamente na primeira descendente não abstracta. Por exemplo os métodos maximoDiasEmprestimo, nomeTipo, utilizadorPodeRequisitar, entre outros seriam definidos como abstractos e não possuiriam código. Assim seriam codificados pelas classes Livro, DVD, Revista, e todos aqueles concretos que herdassem da classe abstracta mas os métodos já codificados na classe abstracta poderiam ser aproveitados, poupando código e trabalho.

```
public abstract class Publicacoes {
    //Variáveis que segundo as regras do encapsula-
mento deverão ser privadas
    public Publicacoes(){
        //Código
    }
    public Publicacoes(String codigo){
        //Código
    }
    public String getCodigo(){
        //Código
        return "x";
    }
  public int getQuantidade(){
        //Código
        return 0;
    }
    //... Mais métodos públicos e se necessário
alguns privados

    public abstract int maximoDiasEmprestimo();

    public abstract String nomeTipo();

    public abstract boolean utilizadorPodeRequisi-
tar(String usercodigo);
}
public class Livro extends Publicacoes{
    public Livro(String codigo){
        //Código
    }
    @Override
    public int maximoDiasEmprestimo(){
        //Aqui já terá código também
        return 7;
    }

    @Override
    public String nomeTipo(){
        //Aqui já terá código  também
        return "Livro";
    }

    @Override
    public boolean utilizadorPodeRequisitar(String
usercodigo){
        //Aqui já terá código  também
        return true;
    }
}
```

Este é um exemplo da aplicação de uma classe abstracta e de uma descendente que implementa as funções abstractas e como tal é possível de criar uma instância do mesmo, onde, tal como na herança entre classes simples, podemos chamar os métodos públicos definidos e herdados da classe mãe.

Diferentes Tipos de Colecções

A JFC (JAVA Collections Framework) oferece ao programador muito mais que simplesmente o ArrayList. Este é apenas uma pequena parte, embora seja talvez a mais utilizada nem sempre adequadamente.

Para um conjunto de problemas que necessitem de uma colecção de valores guardados em memória, muitos programadores por desconhecimento utilizam muitas vezes os métodos menos adequados. A JFC está dividida em três grandes grupos as Listas (List<E>), os Conjuntos (Set<E>) e as Correspondências(Map<K,V>).

Supondo, por exemplo uma lista de amigos, qual seria talvez o grupo mais interessante? Todos eles podem de maneira mais ou menos expedita servir para guardar esta informação. Se a informação fosse guardada na lista teríamos que ter em atenção que não poderíamos repetir amigos e seria necessário andar sempre a verificar a existência o valor a inserir antes de o fazer. Como as listas não são ordenadas por definição, a pesquisa demoraria algum tempo para listas de amigos extensas.

Nas correspondências teríamos o mesmo problema de verificar a existência de amizade, mas teríamos a vantagem de esta estar optimizada para pesquisas. Contudo teríamos a desvantagem de por cada amigo ser necessário guardar um valor, o que pode acabar por não ser uma desvantagem, se for necessário, por exemplo guardar algo associado a essa amizade.

Num conjunto não seria necessária a verificação de repetidos porque ele próprio faz essa verificação, uma vez que a definição matemática de conjunto implica isso mesmo que não haja repetições. É por isso importante estar a par das possibilidades existentes e saber decidir qual a melhor opção.

Conclusão

Tal como disse no inicio este não pretendeu ser um artigo que ensinasse a programar Java, nem POO, mas sim um artigo que mostrasse algumas novidades de Java 6 em relação ao Java 2 (que por motivos de compatibilidade ainda existem em Java 6, mas são desaconselhados) e a sua ligação à Programação Orientada a Objectos, já que é possível programar em Java sem ter em conta os princípios de POO. Apesar de tudo ficou ainda muito por dizer já que Java é uma linguagem muito extensa e foram focados apenas alguns pormenores, no entanto são pormenores que podem permitir um código mais limpo, talvez mais rápido e por ventura mais fácil de actualizar.

Bibliografia

http://wiki.portugal-a-programar.org/dev_geral:java:tutorial:home

http://download.oracle.com/javase/6/docs/api/

AUTOR

António Silva, actualmente a frequentar o 3º ano da Licenciatura de Engenharia Informática da Universidade do Minho sente uma enorme paixão pela programação, nomeadamente em VB.Net e C. Apesar disso possui um conhecimento sobre várias outras linguagens de programação como Java, PHP, Javascript, C#, Haskell, entre outras.

Elege o melhor artigo desta edição

Revista PROGRAMAR

http://tiny.cc/ProgramarED31_V

DataBinding em Silverlight 4

Neste artigo pretendo apresentar o conceito databinding em Silverlight 4. Vou começar por uma breve apresentação teórica e em seguida irei apresentar vários exemplos. De salientar que não terei em conta Design Patterns.

O conceito Databinding permite-nos de uma forma simples e consistente apresentar e interagir a informação da nossa aplicação, através da criação de um relacionamento entre dois objectos para que a alteração de uma propriedade de um deles seja reproduzida numa propriedade de um outro. Existindo uma separação entre a interface com o utilizador e a informação, é estabelecido qual o fluxo entre os dois, a forma como a alteração é reflectida, a forma como os erros são detectados e a forma como a informação é mostrada na interface com o utilizador.

> **Databinding permite-nos de uma forma simples e consistente (…) a criação de um relacionamento entre dois objectos**

A classe **Binding** representa a relação entre a interface com o utilizador e a informação. Esta classe está incluída no **namespace** System.Windows.Data e na **assembly** System. Windows (na System.Windows.dll)

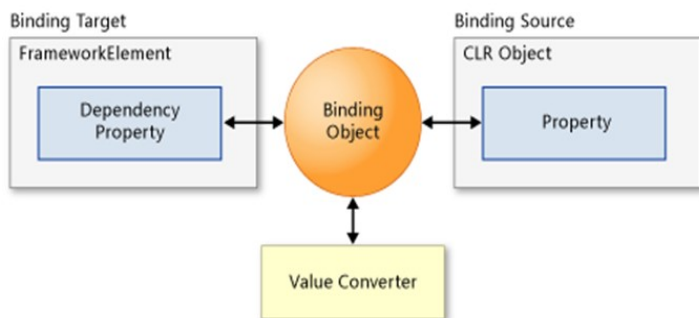



Fig.1 Esquema de relacionamento do Databinding

- **Binding Target** – vai representar um objecto do tipo DependencyObject ou do tipo FrameworkElement, ou seja, na maioria dos casos o controlo da interface com o utilizador.
- **Binding Source** – vai representar um objecto fonte a visualizar/editar;
- **Dependency Property** – permite definir o valor da propriedade e reflectir/propagar a alteração do valor atribuído, assim como permite definir um valor de omissão.
- **Value converter** – é um conversor que permitir efectuar uma transformação

As propriedades a ter em conta:

◊ Converter permite converter a informação apresentar ao utilizador;

◊ ConverterCulture permite definir qual a cultura a utilizar no conversor;

◊ ConverterParameter permite definir o parâmetro do conversor;

◊ ElementName permite definir o nome do controlo ao qual se está usar no binding como objecto fonte;

◊ FallbackValue permite definir qual o valor atribuir em caso de erro;

◊ Mode permite definir o modo do binding; Existem dois tipos: Default e Explicit.

◊ Path permite definir a propriedade a qual se atribui o binding;

◊ RelativeSource permite definir qual o objecto fonte associado, existindo uma hierarquia relativa ao objecto fonte actual;

◊ Source permite definir qual o objecto fonte;

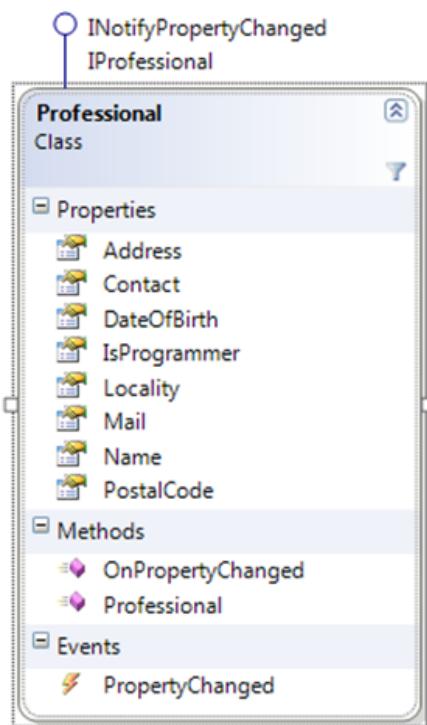
◊ StringFormat permite especificar como formatar a informação a apresentar;

◊ TargetNullValue permite definir o valor a usar quando o objecto fonte é nulo;

◊ UpdateSourceTrigger permite definir o tipo de actualização do binding.

As propriedades ValidatesOnDataErrors; ValidatesOnExceptions; NotifyOnValidationError; ValidatesOnNotifyDataErrors estão relacionadas com validação de dados e notificação de erros. Devem ser definidos a quando da implementação da interface IDataErrorInfo ou INotifyDataErrorInfo.

De seguida iremos apresentar alguns exemplos práticos!

Suponhamos a criação de uma classe chamada **Professional**, que representa um profissional da comunidade NetPonto. Esta classe implementa as interfaces IProfessional, que define o que é um Professional e a interface INotifyPropertyChanged, que permite propagar as alterações que ocorrem.

```
public class Professional : INotifyPropertyChanged,
IProfessional
   {
       #region ------ Private members ------
        private string _name;
        private DateTime _dateOfBirth;
        private string _address;
        private string _locality;
        private int _postalCode;
        private string _mail;
        private int _contact;
        private bool _isProgrammer;

       #endregion
```

```
        public Professional()
        {
            _isProgrammer = true;
        }

        #region ------ Properties ------
        public string Name
        {
            get { return _name; }
            set { _name = value;
                OnPropertyChanged("Name");
            }
        }

        public DateTime DateOfBirth
        {
            get { return _dateOfBirth; }
            set
            {
                _dateOfBirth = value;
                OnPropertyChanged("DateOfBirth");
            }
        }

        public string Address
        {
            get { return _address; }
            set
            {
                _address = value;
                OnPropertyChanged("Adress");
            }
        }

        public string Locality
        {
            get { return _locality; }
            set
            {
                _locality = value;
                OnPropertyChanged("Locality");
            }
        }

        public int PostalCode
        {
            get { return _postalCode; }
            set
            {
                _postalCode = value;
                OnPropertyChanged("PostalCode");
            }
        }

        public int Contact
        {
            get { return _contact; }
```

```
            set
            {
                _contact = value;
                OnPropertyChanged("Contact");
            }
        }


        public string Mail

        {

            get { return _mail; }
            set
            {
                _mail = value;
                OnPropertyChanged("Mail");
            }
        }


        public bool IsProgrammer

        {

            get { return _isProgrammer; }
            set
            {
                _isProgrammer = value;
                OnPropertyChanged("IsProgrammer");

            }
        }
       #endregion


       #region ------ INotifyPropertyChanged ------

        /// <summary>
        /// Called when [property changed].
        /// </summary>
        /// <param name="propertyName">Name of the
property.</param>
        public void OnPropertyChanged(
                              string propertyName)
        {
            if (PropertyChanged != null)
                PropertyChanged(this,
                  new PropertyChangedEventArgs
                              (propertyName));
        }


        public event PropertyChangedEventHandler
                                   PropertyChanged;

        #endregion

    }
```

Nota: Nos próximos casos práticos o source é atribuído no DataContext da Page e o binding irá subir na hierarquia até encontrar o source.

Caso prático 1

Suponhamos que pretendemos interagir a propriedade Name com uma TextBox.

Em XAML:

```
<TextBox Name="tbxName"
         Text="{Binding Path=Name,
         Mode=TwoWay,
         UpdateSourceTrigger=Default}"/>
```

Ou em *code-behind*:

```
var binding =new Binding("Name")

  {

    Mode = BindingMode.TwoWay, _
      UpdateSourceTrigger _
      =UpdateSourceTrigger.Default

   };

tbxName.SetBinding(TextBox.TextProperty, _

binding);
```

Caso prático 2

Suponhamos que pretendemos interagir a propriedade IsProgrammer com uma CheckBox.

Em XAML:

```
<CheckBox Name="ckboxIsProgrammed"
IsChecked="{Binding Path=IsProgrammer,
Mode=TwoWay,UpdateSourceTrigger=Default}"/>
```

Ou em *code-behind*:

```
var binding = new Binding("IsProgrammer")

{

Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.Default

 };


ckboxIsProgrammed.SetBinding
(CheckBox.IsCheckedProperty, binding);
```

Caso prático 3

Suponhamos que pretendemos apresentar o PostalCode numa label, usando um conversor para formatar a informação.

Como o PostalCode é um valor inteiro que guarda o código postal, o PostalCodeConverter é o conversor que separa os primeiros 4 dígitos dos 3 últimos dígitos usando um hífen.

A classe PostalCodeConverter implementa a interface IValueConverter e consiste na implementação de dois métodos fundamentais, o Convert e o ConvertBack.

```
public class PostalCodeConverter:IValueConverter
{
public object Convert(object value, System.Type
targetType, object parameter, System.Globalization
.CultureInfo culture)
{
 return value.ToString().Substring(0, 4) + "-" +
value.ToString().Substring(4, 3);
}
public object ConvertBack(object value,
System.Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
  return int.Parse(value.ToString().Remove('-'));
}
}
```

No XAML:

É preciso definir o namespace

```
xmlns:sdk="http://schemas.microsoft.com/
winfx/2006/xaml/presentation/sdk"
```

É preciso definir o conversor nos resources:

```
<navigation:Page.Resources>
   <Demo:PostalCodeConverter
                x:Key="postalCodeConverter" />
</navigation:Page.Resources>
```

Definição da label

```
<sdk:Label Name="lblPostalCode"
      Content="{Binding Path=PostalCode,
            Mode=TwoWay,
            UpdateSourceTrigger=Default,
            Converter={StaticResource
                  postalCodeConverter}}" />
```

Caso prático 4

Atribuir uma lista de profissionais a uma listbox, apresentando a propriedade Name.

No XAML:

```
<ListBox Name="lbxProfessional"
         DisplayMemberPath="Name"/>
```

Em code behind:

```
var professionals = new List<Professional>();
...
lbxProfessional.ItemsSource = professionals;
```

Caso prático 5

Atribuir uma lista de profissionais a uma datagrid.

No XAML:

É preciso definir o namespace

```
xmlns:sdk="http://schemas.microsoft.com/
winfx/2006/xaml/presentation/sdk"
<sdk:DataGrid x:Name="datagrid"
            AutoGenerateColumns="False">
<sdk:DataGrid.Columns>
 <sdk:DataGridTextColumn Header="Nome"
      MinWidth="200"
      Binding="{Binding Name}" />
 <sdk:DataGridTextColumn
      Header="Código Postal"
      Width="100"
      MaxWidth="150"
      Binding="{Binding Path=PostalCode,
            Converter={StaticResource
                  postalCodeConverter}}"/>
 <sdk:DataGridTextColumn
      Header="Endereço"
      Width="50"
      Binding="{Binding Path=Address}"/>
 <sdk:DataGridTextColumn
      Header="Localidade"
      Binding="{Binding Path=Locality}"/>

</sdk:DataGrid.Columns>

</sdk:DataGrid>
```

Em *code-behind*:

```
datagrid.ItemsSource = professionals;
```

Caso prático 6

Definir um ItemTemplate para uma listbox.

No XAML:

```
<ListBox Name="lbxProfessionalWidthDataTemplate">
  <ListBox.ItemTemplate>
   <DataTemplate>
    <StackPanel Orientation="Vertical">
      <TextBlock Text="{Binding Path=Name}"/>
      <TextBlock Text="{Binding _
            Path=PostalCode,
            Converter={StaticResource
                        postalCodeConverter}}"/>
      <TextBlock Text="{Binding Path=Address}"/>
      <TextBlock Text="{Binding Path=Locality}"/>
      <TextBlock Text="{Binding Path=Contact}"/>
      <TextBlock Text="{Binding Path=Mail}"/>
    </StackPanel>
   </DataTemplate>
  </ListBox.ItemTemplate>
 </ListBox>
```

Caso prático 7

Formatar o dia de aniversário usando o StringFormat.

No XAML:

```
<TextBox Name="tbxDateOfBirth" Grid.Column="2"
Grid.Row="2" Text="{Binding Path=DateOfBirth,
StringFormat=MM-dd-yyyy}"/>
```

Ou

```
<TextBox Name="tbxDateOfBirth" Grid.Column="2"
Grid.Row="2" Text="{Binding Path=DateOfBirth,
StringFormat=MMM dd yyyy}"/>
```

Conclusão

Em conclusão, o conceito Databinding é uma forma simples e eficiente de visualização e edição da informação nas aplicações desenvolvidas em Silverlight.

AUTOR

Sara Silva, é licenciada em Matemática – Especialidade em Computação, pela Universidade de Coimbra, actualmente é Software Developer no Porto.

O entusiasmo pela área resultou na obtenção dos títulos de *Microsoft Certified Profissional Developer – Windows 3.5*, *Microsoft Certified Tecnology Specialist – WPF 3.5, WPF 4 e Windows Forms*.

COLUNAS

VISUAL (NOT) BASIC – Primeiros passos com GDI+

CoreDump – Martelo <=> Inépcia

Primeiros passos com GDI+

GDI?

GDI ou Graphic Device Interface é a API responsável, directa ou indirectamente, total ou parcialmente, por tudo o que nos chega aos olhos no mundo Windows.

Se não existe aceleração por hardware e estamos a ver curvas, linhas, texto e gráficos desenhados, é porque a GDI assim o permite.

Através desta API conseguimos, de forma bastante penosa (para os produtos de desenvolvimento Microsoft de outrora), desenhar geometria, gráficos e fontes para o ecrã ou para a impressora.

Com a evolução dos tempos, surgem novas necessidades e por volta do Windows XP a Microsoft cria uma API para C/C++ que chamou simplesmente de GDI+.

A API resistiu até hoje, onde já se tentam incutir tecnologias baseadas em GDI mas que visam substituí-la, como Direct2D.

A nova API reduziu significativamente a complexidade para o programador.

GDI na .NET

A API já se tinha tornado mais poderosa, mais fácil e representando uma bela fatia no bolo da relevância de um sistema operativo, só seria natural criar um acesso de alto nível, muito mais fácil e controlado na .NET framework. De facto a Microsoft não poderia ter tornado o *namespace* mais simples.

Para ganharmos acesso a todas as classes expostas basta-nos importar / fazer referência ao *namespace* **System.Drawing**.

É neste *namespace* que se encontram todas as classes e *sub-namespaces* que nos permitem trabalhar imediatamente com a API.

Onde posso desenhar?

Como todos os componentes passam pela GDI, todos representam uma potencial folha em branco para os nossos dotes artísticos.

Isto significa que todos os exemplos neste artigo, se aplicados por exemplo num form, podem perfeitamente ser aplicados desde o panel até a um item de listbox.

A classe Graphics é a nossa porta da frente.

Podemos preparar uma superfície GDI+ com muita facilidade.

No decorrer do artigo, vou desenhar tudo num *panel*, aproveitando o disparo do evento “Paint”, que ocorre sempre que a área de desenho precise de ser actualizada, quer por indicação explícita invalidando o interface do objecto - PanelGDI.Invalidate() ,ou porque a área saiu do ecrã ou tem outra janela a tapar.

Normalmente podemos referenciar a superfície GDI do objecto por:

```
Dim GDI As Graphics = PanelGDI.CreateGraphics()
```

Neste caso, como vamos utilizar o disparo do evento Paint, um dos parâmetros que nos chegam do evento é precisamente uma referência à superfície GDI:

```
Dim GDI As Graphics = e.Graphics
```

Preparar para os primeiros rabiscos

Com a superfície GDI preparada, poderíamos começar a desenhar imediatamente.

Existem essencialmente 4 maneiras diferentes para “desenhar”:

Bitmap, Pen, Brush e Font, servem respectivamente para desenhar imagens, desenhar linhas, pintar áreas e desenhar letras.

Diferentes métodos utilizam diferentes formas de desenhar e todos em conjunto formam composições.

Estas formas de desenhar têm que ser preparadas antes de poderem ser utilizadas, mas antes…

Tamanho, posição e cor

Antes de poder começar a desenhar é importante entender algumas classes que nos ajudam a posicionar, dimensionar e colorir os nossos rabiscos:

Point / PointF

O Point é uma estrutura que representa essencialmente um conjunto de dois valores, X e Y, que representam uma coordenada num referencial bidimensional.

Podemos criar um ponto directamente no construtor:

```
Dim Ponto As New Point(25, 35)
Dim PontoF As New PointF(20.5F, 22.03F)
```

Size / SizeF

O Size é uma estrutura que representa um conjunto de dois valores, Width e Height, que representam valores de largura e altura, respectivamente.

Podemos criar uma dimensão directamente no construtor:

```
Dim Dimensao As New Size(800, 600)
Dim DimensaoF As New SizeF(100.05F, 100.07F)
```

Rectangle / RectangleF

O Rectangle é uma estrutura que representa um conjunto de 4 valores, X, Y, Width e Height, que representam um combinado de coordenadas XY e dimensão.

Podemos criar um Rectangle directamente no construtor:

Color

```
Dim Rectangulo As New Rectangle(0, 0, 100, 100)
Dim RectanguloF As _
    New RectangleF(0.5F, 0.8F, 100.03F, 100.25F)
```

A Color é uma estrutura que representa um conjunto de 4 valores, A, R, G e B que representam os 4 canais que constituem uma cor aRGB: A para Alpha, ou o canal de opacidade, RG e B para Red, Green e Blue, respectivamente.

Cada canal comporta valores que variam dos 0 (ausência) até 255 (presença total)

A classe Color não tem construtores. Podemos obter uma cor quer através da enumeração disponível, quer por utilização de alguns métodos disponíveis.

```
Dim Cor As Color = Color.FromArgb(255, 255, 0, 0)
Dim Cor As Color = Color.Red
Dim Cor As Color = Color.FromName("Red")
Dim Cor As Color = _
    Color.FromKnownColor(KnownColor.ActiveBorder)
```

Com estas noções estamos aptos a colorir e posicionar os nossos rabiscos.

As principais formas de desenhar

Bitmap / Image

A classe Bitmap representa uma imagem completa, com todas as suas propriedades e informação de cor pixel a pixel.

Nos métodos que impliquem desenhar imagens, é uma instância desta classe que vai ser usada.

Bitmap tem vários construtores, bastante específicos, mas vamos apenas considerar dois para este artigo.

```
Dim Imagem As New Bitmap("C:\imagem.png")
Dim Imagem As New Bitmap(800, 600)
```

O primeiro construtor vai instanciar a classe com a informação da imagem indicada.

O segundo construtor vai criar uma imagem abstracta, com 800x600 px

O primeiro construtor vai instanciar a classe com a informação da imagem indicada.

O segundo construtor vai criar uma imagem abstracta, com 800x600 px

Pen

A Pen é necessária para todos os métodos que impliquem o desenho de linhas, curvas e contornos.

Podemos criar uma Pen apenas por indicar a sua cor e espessura:

Mas existem mais propriedades na classe Pen que lhe

```
Dim P As New Pen(Color.Black,2)
```

podem conferir mais características visíveis, tais como a forma terminal ou o estilo da linha, que não vou abordar.

Brush

O Brush é necessário para todos os métodos que impliquem preenchimento de áreas.

Existem vários tipos de “pincel” que permitem a aplicação de texturas, gradientes, padrões ou simplesmente uma cor.

Vou focar essencialmente o SolidBrush, para cores únicas e o LinearGradientBrush para gradientes de duas cores simples.

Podemos criar SolidBrush directamente no construtor:

```
Dim B As New SolidBrush(Color.Red)
```

Também podemos criar LinearGradientBrush directamente no construtor:

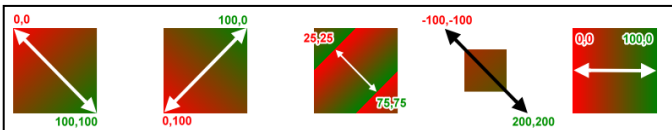
```
Dim B As New LinearGradientBrush(New  _
Point(0, 0), New  _
Point(100, 100), Color.Red, Color.Green)
```

Primeiros passos com GDI+

É necessário um ponto onde o gradiente vai iniciar e um ponto onde vai terminar, bem como as cores nesses mesmos pontos.

A mistura de cor no intervalo intermédio é automaticamente calculada.

Existem classes que desempenham um papel importante na mistura de cores do gradiente, mas não as vamos abordar.

Observemos diferentes colocações dos pontos do gradiente abaixo:

Font

A Font é necessária para todos os métodos que impliquem o uso das fontes do sistema.

Será tipicamente utilizada sempre que for necessário escrever alguma coisa na superfície GDI.

Podemos criar uma Font directamente no construtor:

```
Dim F As New Font("Arial", 14)
Dim F As New Font("Arial", 14, FontStyle.Bold Or _
    FontStyle.Italic)
Dim F As New Font("Arial", 14, FontStyle.Bold Or _
    FontStyle.Italic, GraphicsUnit.Pixel)
```

Os métodos, finalmente.

Descrita toda a preparação, está na altura de explorar os métodos da classe Graphics.

O namespace "Drawing" é vasto e bastante completo.

Vou apontar apenas o que considero mais importante.

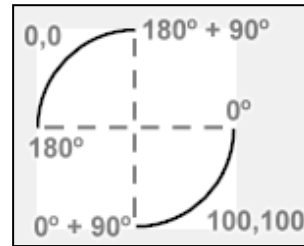
DrawArc(Pen,X,Y,Width,Height,AnguloInicial,Varrimento)

O método DrawArc desenha um arco com as propriedades da Pen, com centro em XY, incluído no rectângulo de largura Width e altura Height, centrado.

A curva começa no AnguloInicial e é desenhada até ao ângulo AnguloInicial + Varrimento.

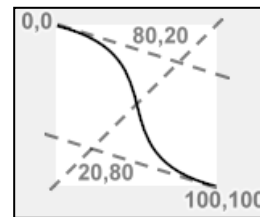
Os ângulos são dados em graus.

Existem mais overloads, mas são redundantes.

```
Dim g As Graphics = e.Graphics
g.DrawArc(New Pen(Brushes.Black, 2), 0, 0, 100,
100, 0, 90)
g.DrawArc(New Pen(Brushes.Black, 2), 0, 0, 100,
100, 180, 90)
```

DrawBezier (Pen, Ponto1,Ponto2,Ponto3,Ponto4)

O método DrawBezier desenha uma curva de Bézier com 4 pontos de controlo.

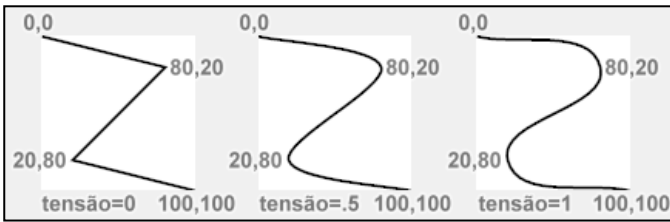
Existem mais overloads, mas são redundantes.

```
Dim pt1 As New Point(0, 0)
Dim pt2 As New Point(80, 20)
Dim pt3 As New Point(20, 80)
Dim pt4 As New Point(100, 100)
Dim g As Graphics = e.Graphics
g.DrawBezier(New Pen(Brushes.Black, 2), _
pt1, pt2, pt3, pt4)
```

DrawCurve / DrawColsedCurve (Pen,Pontos(),Tensão)

O método DrawCurve desenha uma curva linear, passando por os pontos no array Pontos(), aplicando a tensão defenida (valor de 0.0 a 1.0)

DrawClosedCurve é em tudo semelhante, mas liga o último ponto em Pontos() ao primeiro, fechando a curva.

Existem mais overloads.

```
Dim pt1 As New Point(0, 0)
Dim pt2 As New Point(80, 20)
Dim pt3 As New Point(20, 80)
Dim pt4 As New Point(100, 100)
Dim pts As Point() = New Point() {pt1, pt2, pt3,
pt4}
Dim g As Graphics = e.Graphics
g.DrawCurve(New Pen(Brushes.Black, 2), pts, 0F)
g.DrawCurve(New Pen(Brushes.Black, 2), pts, 0.5F)
g.DrawCurve(New Pen(Brushes.Black, 2), pts, 1F)
```

DrawEllipse (Pen, Rectangulo)

O método DrawEllipse desenha uma elipse inscrita no rectângulo especificado.

A elipse é desenhada no centro do rectângulo o que faz com que se possa afirmar que, por exemplo, num quadrado 100x100, o método DrawEllipse vai desenhar uma circunferência centrada em x=50 e y=50 com raio de 50px

```
Dim rect1 As New Rectangle(0, 0, 100, 100)
Dim rect2 As New Rectangle(0, 0, 100, 50)
Dim g As Graphics = e.Graphics
g.DrawEllipse(New Pen(Brushes.Black, 2), rect1)
g.DrawEllipse(New Pen(Brushes.Black, 2), rect2)
```

DrawImage (Bitmap,Ponto / Rectangulo)

O método DrawImage desenha a informação de uma classe Bitmap com origem no ponto especificado ou com origem e dimensões ditadas pelo rectângulo.

Existem outros métodos semelhantes e muitos overloads, a maioria redundantes.

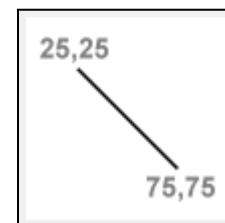
O canal Alpha é perfeitamente respeitado e obtêm-se bonitos resultados com PNG.

```
Dim B As New Bitmap("c:\pap.png")
Dim Ponto As New Point(0, 0)
Dim g As Graphics = e.Graphics
g.DrawImage(B, Ponto)
Dim Rectangulo As New Rectangle(0, 50, 50, 50)
g.DrawImage(B, Rectangulo)
```

DrawLine (Pen,Ponto1,Ponto2)

O método DrawLine desenha uma linha com origem em Ponto1 e destino em Ponto2.

Existem mais overloads, mas são redundantes.

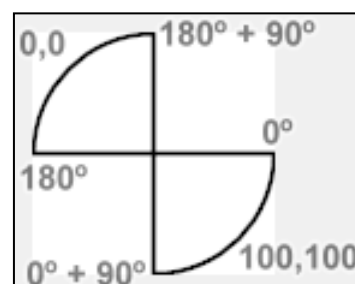
```
Dim pt1 As New Point(25, 25)
Dim pt2 As New Point(75, 75)
Dim g As Graphics = e.Graphics
g.DrawLine(New Pen(Brushes.Black, 2), pt1, pt2)
```

DrawPie (Pen,X,Y,Width,Height,AnguloInicial,Varrimento)

O método DrawPie desenha uma secção de um círculo, lembrando uma fatia de pizza ou tarte (Pie).

A circunferência é inscrita no rectângulo formado por XY, Width e Height e é desenhada a secção da circunferência que começa no AnguloInicial e termina em AnguloInicial+Varrimento.

Os ângulos são dados em graus.

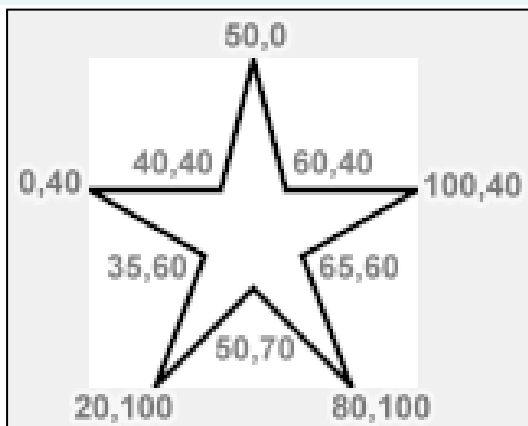
Existem mais overloads, mas são redundantes.

```
Dim g As Graphics = e.Graphics
g.DrawPie(New Pen(Brushes.Black, 2), _
    0, 0, 100, 100, 0, 90)
g.DrawPie(New Pen(Brushes.Black, 2), _
    0, 0, 100, 100, 180, 90)
```

Primeiros passos com GDI+

DrawPolygon (Pen,Pontos())

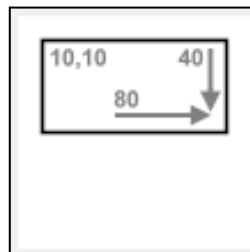
O método DrawPolygon desenha um polígono, ao ligar sequencialmente todos os pontos especificados, voltando ao primeiro depois do último.

Existe apenas mais um overload para utilizar valores com casas decimais.

```
Dim pt1 As New Point(50, 0)
Dim pt2 As New Point(60, 40)
Dim pt3 As New Point(100, 40)
Dim pt4 As New Point(65, 60)
Dim pt5 As New Point(80, 100)
Dim pt6 As New Point(50, 70)
Dim pt7 As New Point(20, 100)
Dim pt8 As New Point(35, 60)
Dim pt9 As New Point(0, 40)
Dim pt10 As New Point(40, 40)
Dim pts As Point() = New Point() _
    {pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8, pt9, pt10}
Dim g As Graphics = e.Graphics
g.DrawPolygon(New Pen(Brushes.Black, 2), pts)
```

DrawRectangle (Pen,X,Y,Width.Height)

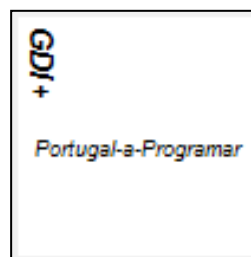
O método DrawRectangle desenha um rectângulo com origem no XY especificado e com as dimensões especificadas.

Existem mais overloads, mas são redundantes.

```
Dim g As Graphics = e.Graphics
g.DrawRectangle(New _
    Pen(Brushes.Black, 2), 10, 10, 80, 40)
```

DrawString (Texto,Fonte,Brush,Rectangulo,StringFormat)

O método DrawString desenha texto a partir de uma string, com uma determinada fonte e conjunto de características de formato.

Existem overloads, alguns redundantes.

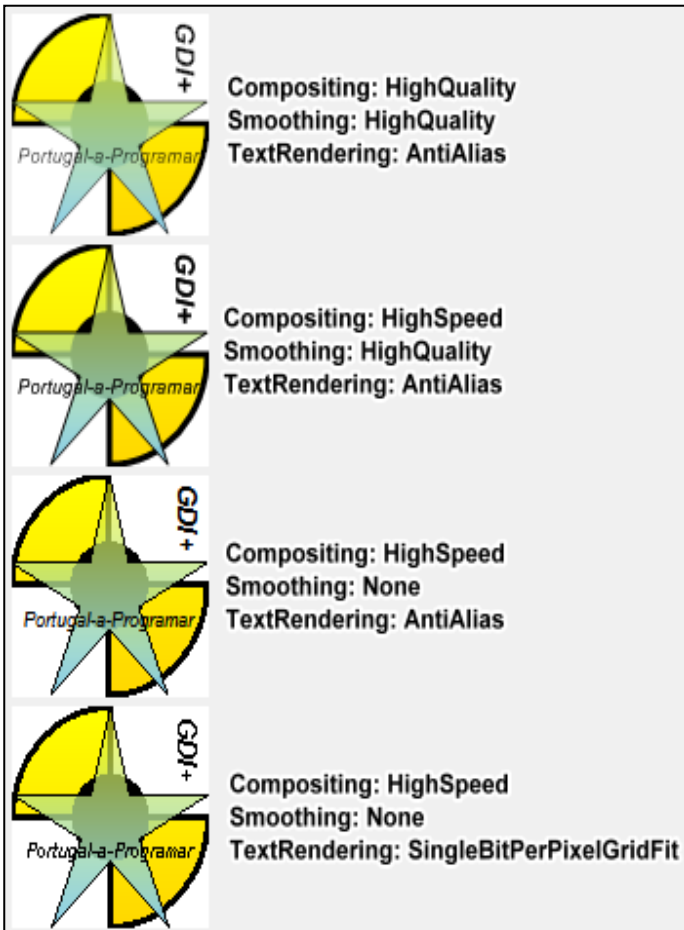
```
Dim F As New Font("Arial", 7, FontStyle.Italic)
Dim SF As New StringFormat
SF.Alignment = StringAlignment.Center
Dim R As New Rectangle(0, 50, 100, 40)
Dim g As Graphics = e.Graphics
g.DrawString("Portugal-a-Programar", F, _
    Brushes.Black, R, SF)
SF.FormatFlags = StringFormat-Flags.DirectionVertical _
    Or StringFormatFlags.LineLimit
F = New Font("Arial", 10, FontStyle.Italic _
    Or FontStyle.Bold)
g.DrawString("GDI+", F, Brushes.Black, 0, 20, SF)
```

A maioria dos métodos de prefixo "Draw" existe numa segunda versão "Fill".

A operação é semelhante, mas as áreas geradas são preenchidas de acordo com um brush fornecido, quer seja ele sólido, gradiente, textura, hatch ou de qualquer outro tipo.

Performance versus Qualidade

A classe Graphics possui algumas propriedades que definem a forma como a GDI vai render os gráficos.

Como em tudo, se pretendemos qualidade temos de sacrificar performance e vice-versa.

Eis algumas das propriedades mais comuns e seus efeitos:

Algumas dicas

Double Buffer

Se a intenção é usar a GDI+ para criar algum tipo de movimento, tal como um pequeno jogo, algo que necessite de constante e rápido refrescamento, rapidamente se nota que se torna impossível de continuar pois cada ciclo de refrescamento é reflectido de forma negativa na superfície GDI.

Isto traduz-se num "piscar" insuportável que é causado pela constante soma de execução do refrescamento e dos gráficos propriamente ditos.

Nestes casos podemos, sacrificando um pouco de performance, utilizar um "Double buffer" que fará com que as actualizações sejam feitas em paralelo com a apresentação, o que se traduz numa actualização "suave" da superfície GDI, mas um pouco mais demorada.

Num objecto como a Form, basta alimentar a propriedade **DoubleBuffered** para **True.**

MeasureString

Nos casos em que desenhamos texto indicando apenas uma posição de origem XY, este é desenhado de forma linear, sem qualquer limite.

O método MeasureString permite-nos medir o tamanho que essa string vai ocupar, para que possamos fazer correctamente os nossos cálculos.

Também é possível medir as dimensões de uma string desenhada dentro de uma área, que faz com que possamos ter noção do quanto "cresceu" nos casos onde ocorre "wordwrap".

O método MeasureString é muito semelhante ao método DrawString, pois precisa de saber exactamente como é que o texto foi (ou vai ser) desenhado, para que possa internamente fazer a simulação.

O método devolve-nos as dimensões do texto, em largura e altura, num tipo de dados SizeF.

Pen Cap / Dash Style

Ao desenhar linhas, sobretudo com alguma espessura visível, podemos notar que os terminais da linha são abruptamente cortados, como é normal.

O pen cap determina precisamente como esses terminais são desenhados.

Existem alguns caps disponíveis, para além da possibilidade de desenhar os nossos próprios caps (custom).

Para além dos caps e de outras propiedades, é possível determinar como vai ser o traçado da linha, o Dash Style.

Para este exemplo foi necessário utilizar a classe GraphicsPath, a qual não está prevista para este artigo.

```
Dim g1 As Graphics = PanelGDI.CreateGraphics
Dim g2 As Graphics = Panel1.CreateGraphics
Dim g3 As Graphics = Panel2.CreateGraphics
Dim g4 As Graphics = Panel3.CreateGraphics
Dim g5 As Graphics = Panel4.CreateGraphics
g1.Clear(Color.White)
g2.Clear(Color.White)
g3.Clear(Color.White)
g4.Clear(Color.White)
g5.Clear(Color.White)
```

```
Dim pt1 As New Point(20, 20)
Dim pt2 As New Point(80, 20)
Dim pt3 As New Point(20, 80)
Dim pt4 As New Point(80, 80)
Dim MyP As New Pen(Brushes.Black, 6)
g1.DrawBezier(MyP, pt1, pt2, pt3, pt4) 'IMG1

MyP = New Pen(Brushes.Black, 6)
MyP.StartCap = LineCap.DiamondAnchor
MyP.EndCap = LineCap.ArrowAnchor
g2.DrawBezier(MyP, pt1, pt2, pt3, pt4) 'IMG2
MyP = New Pen(Brushes.Black, 6)
MyP.DashStyle = DashStyle.Custom
MyP.DashPattern = New Single() {0.5F, 1.0F, 0.5F,
1.0F}
Dim Seta As New AdjustableArrowCap(1, 2)
Seta.Filled = False
MyP.CustomEndCap = Seta
g3.DrawBezier(MyP, pt1, pt2, pt3, pt4) 'IMG3
MyP = New Pen(Brushes.Black, 6)
MyP.DashCap = DashCap.Round
MyP.EndCap = LineCap.RoundAnchor
g4.DrawBezier(MyP, pt1, pt2, pt3, pt4) 'IMG4
MyP = New Pen(Brushes.Black, 1.5F)
Dim GP As New GraphicsPath()
Dim cpt1 As New Point(5, 0)
Dim cpt2 As New Point(6, 4)
Dim cpt3 As New Point(10, 4)
Dim cpt4 As New Point(6, 6)
Dim cpt5 As New Point(8, 10)
Dim cpt6 As New Point(5, 7)
Dim cpt7 As New Point(2, 10)
Dim cpt8 As New Point(3, 6)
Dim cpt9 As New Point(0, 4)
```

```
Dim cpts As Point() = New Point() _
    {cpt1, cpt2, cpt3, cpt4, cpt5, cpt6, cpt7,
cpt8, cpt9, cpt10}
GP.AddPolygon(cpts)
Dim MyC As New CustomLineCap(Nothing, GP)
MyP.EndCap = LineCap.Custom
MyP.StartCap = LineCap.Custom
MyP.CustomStartCap = MyC
MyP.CustomEndCap = MyC
g5.DrawBezier(MyP, pt1, pt2, pt3, pt4) 'IMG5
```

I

mpressão

Dentro do namespace Drawing, pertencente ao namespace Printing, encontramos algumas classes como PrintDocument, que muitos já conhecem para fazer impressão de dados.

Não é uma falha o namespace Printing estar incluído no namespace Drawing.

Na verdade, é possível desenhar documentos para a impressora exactamente da mesma forma que os desenharíamos no form.

Em suma...

O namespace Drawing e a GDI+ no geral, oferecem-nos um grande potencial para a imaginação, a partir do qual podemos criar aplicações visualmente ricas.

A forma como se programa com a API, em nível mais elevado com o namespace Drawing concede uma simplicidade de tal forma que é possível desenhar uma linha com duas linhas de código.

Se queremos enriquecer visualmente uma aplicação WinForms, nem vale a pena procurar mais. A GDI+ é ideal e está incluída na .NET.

AUTOR

Sérgio Ribeiro, curioso e auto-didacta com uma enorme paixão por tecnologias de informação e uma saudável relação com a .NET framework.
Moderador do quadro de Visual Basic.NET na comunidade Portugal@Programar desde Setembro de 2009.
Alguns frutos do seu trabalho podem ser encontrados em http://www.sergioribeiro.com

Martelo <=> Inépcia

Esta semana conversava com um colega e amigo sobre o facto da maioria dos profissionais de TI que chegam actualmente ao mercado de trabalho não compreenderem as bases sobre as quais assentam as tecnologias que utilizam. Pode parecer conversa de "velho do restelo", de quem tinha de saber configurar um disco rígido como *master* ou *slave* através de jumpers no hardware ou de quem tinha de saber configurar o DOS correctamente para poder fazer uso total dos 4 MB de RAM que estavam disponíveis e libertar ao máximo os míticos 640Kb. Acreditem que nenhuma destas experiências é saudosista e que o tema da conversa não originou daí...

A verdade é que **actualmente detecto falhas nas bases de muitos profissionais de TI** que, ou por não terem sido bem orientados, ou pela actual facilidade tecnológica, ou por simples preguiça pessoal, não compreendem na totalidade as tecnologias com que trabalham diariamente.

São exemplos típicos a utilização de frameworks, que facilitam o desenvolvimento através da elevação de conceitos e abstracção, a utilização de bases de dados e a utilização do paradigma Object Oriented.

Uma framework simplifica e agiliza o desenvolvimento, mas também esconde e abstrai o trabalho que está por detrás, fazendo querer a muitos dos seus utilizadores que é magia. Como resultado, temos muitas vezes profissionais de TI incapazes de compreender a sua ferramenta de trabalho, ficando à sua mercê. Este desconhecimento torna-se relevante quando se deparam com determinados bugs relacionados com a framework ou quando necessitam de implementar uma solução que sai fora do âmbito da aplicação da framework.

É preocupante observar que todos os problemas de lentidão de uma aplicação se resolvem com mais RAM ou mais CPU sem sequer se identificar a causa do problema recorrendo à utilização de um profiler.

Uma base de dados é imprescindível para qualquer sistema informático, e desconhecer a 3ª Forma Normal é meio caminho andado para um sistema deficiente. Este problema é visível na nossa própria comunidade, onde recorrentemente se vêm estagiários e estudantes – futuros profissionais de TI – a desenvolverem projectos em que criam tabelas sem saber o que representam e não terem qualquer noção das suas relações. É gritante observar que a única solução para uma query lenta passa pela criação de um índice, quando a primeira abordagem deve ser sempre a observação do plano de execução e a reescrita da query de forma mais eficiente.

E quando chegamos aos paradigmas, enfim... Constato que muitos não sabem sequer o que é a herança no paradigma OO, fazendo classes que diferem apenas num pequeno conjunto de atributos distinto. Pior, ignoram mecanismos como cálculo lambda e paradigmas como os da programação funcional.

Tenho outro amigo que costuma dizer, em relação a este tipo de comportamento padrão, que "***quando a única ferramenta que se conhece é o martelo, todos os problemas parecem um prego***". E a verdade é que esta frase resume um pouco o que se passa. Em vez de se compreender o que se passa para procurar a melhor solução a aplicar, martela-se a única solução que se conhece... Por vezes a aplicação da solução escala com o problema e quanto maior o problema, maior a martelada... Fico a pensar se será ironia ou pura casualidade o facto de tantas aplicações terem um martelo como ícone em algumas das suas opções de menu.

É óbvio que compreender as bases da tecnologia é imprescindível para saber o que andamos a fazer. Se não compreendermos as bases da tecnologia, dificilmente saberemos o que estamos a fazer e se não sabemos o que estamos a fazer, então o resultado do nosso trabalho nunca será grande coisa... Ninguém nasce ensinado, mas nunca na história da Humanidade foi tão simples, rápido e barato aceder ao conhecimento. Na área de TI não faltam livros, blogs, fóruns e afins onde tudo se encontra documentado e descrito para nossa compreensão. Infelizmente, também aqui, vejo que **muitos usam a internet de forma errada, fazendo *copy-paste programming* em vez de compreenderem a solução** e, quando o código copiado de um qualquer site não funciona à primeira, o resultado é um post com um pedido de ajuda. Infelizmente, a nossa comunidade também tem exemplos de sobra deste tipo… A culpa da formação inadequada só é responsável até certo ponto, uma vez que todos estes problemas se resolvem com a adopção de uma postura e atitude de querer saber e compreender por parte de cada um. Por isso peço a todos os que pretendem trabalhar em TI que façam um favor a vocês próprios: larguem o martelo, **continuem a aprender e não cedam à inépcia**.

AUTOR

Fernando Martins, faz parte da geração que se iniciou nos ZX Spectrum 48K. Tem um Mestrado em Informática e mais de uma década de experiência profissional nas áreas de Tecnologias e Sistemas de Informação. Criou a sua própria consultora sendo a sua especialidade a migração de dados.

COMUNIDADES

SharePointPT - Padrão alternativo de Sharepoint

Padrão alternativo de Sharepoint: a pesquisa como fonte de dados – parte I – Arquitectura

A plataforma Sharepoint tem sido nos últimos anos, um importante suporte para o desenvolvimento de soluções empresariais de intra/extra/internet. Com maior ou menor dificuldade as equipas entram no padrão de desenvolvimento modular (p.e. webparts) ou via extensão da própria plataforma, suportado por pacotes de instalação (wsps) e funcionalidades empacotadas (features), tendo asseguradas de base algumas funcionalidades interessantes, como p.e.:

- Controlo de acessos
- Motor de workflows
- Plataforma de edição de conteúdos
- Serviços transversais (pesquisa, excel, business intelligence, etc)
- Instalação de pacotes em toda a farm de forma automática

Neste artigo pretende-se desconstruir as abordagens mais tradicionais, apresentando uma solução pouco comum através de uma arquitectura, mas que optimiza os módulos de maior interesse para o cenário apresentado, centrando-se principalmente na componente de indexação e pesquisa do Sharepoint. Este cenário corresponde a parte de uma implementação real.

Neste 1º artigo, é abordada a componente de arquitectura da solução e componentes de alto nível, para permitir a visualização da solução.

No 2º artigo da série, será descrita a componente mais técnica, incluindo a configuração das regras de pesquisa, das propriedades (crawlled e managed), models de consumo dos dados e o respectivo empacotamento da solução de indexação.

Cenário

A empresa XPTO necessita de um portal público que disponibilize uma loja de comércio electrónico, tendo como principais requisitos:

- Performance: carregamento de páginas até 3 segundos, independentemente do volume de acessos;
- Escalabilidade: a plataforma deve permitir um processo evolutivo simples e sólido;
- Robustez: os tempos de indisponibilidade são críticos para a organização; dada a especialidade do negócio a suportar pela plataforma, a aplicação não deverá apresentar picos diários de utilização, prevendo-se um volume de utilizadores estável no período 06:00 até às 02:00, altura em que a utilização decai para um valor residual;
- Não perder 1 única encomenda: a plataforma deverá permitir recuperar todas as encomendas que possam falhar, para posterior processamento por operadores;
- Ter um portal para gerir a informação genérica dos produtos.

A XPTO tem um volume de aproximadamente 20.000 utilizadores registados, com um acesso diário previsto de 10%.

Arquitectura

Como componentes da arquitectura, é proposta uma abordagem distribuída em vários componentes/plataformas, tirando proveito das principais características de cada uma:

- Apresentação: IIS (Internet Information Services), .Net 4 e MVC
- Como principais vantagens destas tecnologias temos a escalabilidade, robustez e flexibilidade
- Gestão de conteúdos e indexação: Sharepoint
- A gestão de conteúdos e indexação são alguns dos principais pontos fortes da plataforma Sharepoint
- Gestão de catálogo: Commerce Server
- A plataforma de comércio electrónico centra-se

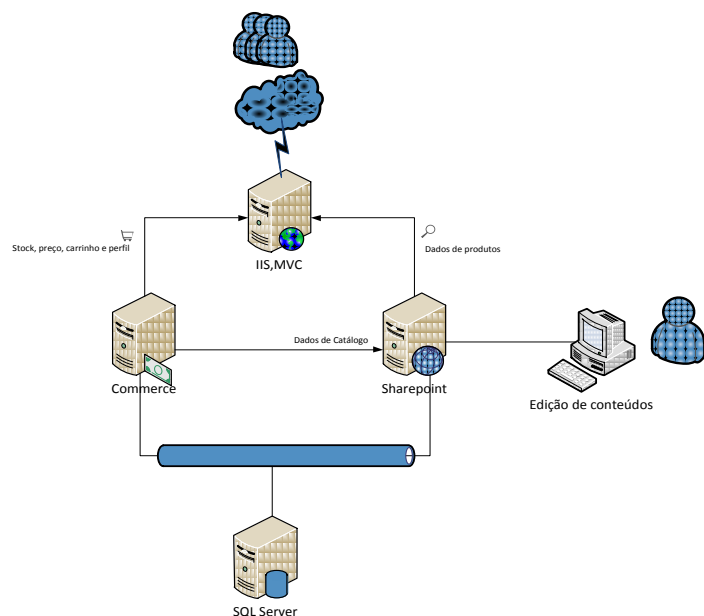
Esta arquitectura conta ainda com as plataformas Biztalk na componente de integração e com o CRM na componente de gestão de clientes, ficando no entanto de fora do tema proposto neste artigo.

A camada de apresentação é integralmente suportada por uma implementação MVC, cujos models vão beber dados principalmente às plataformas Commerce e Sharepoint (search services).

A gestão de conteúdos é suportada por um portal Sharepoint, onde as características de produtos, ficheiros

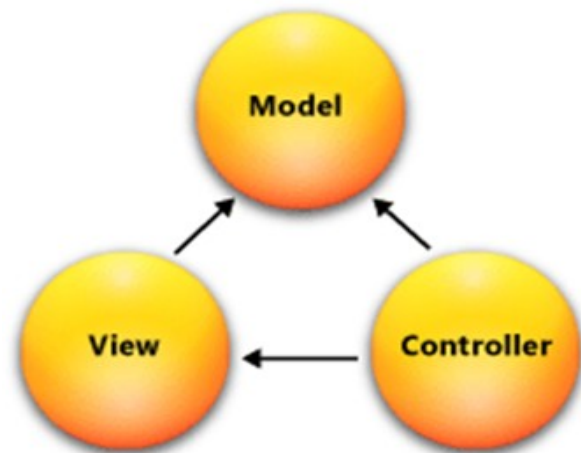
multimédia e documentos são geridos e mantidos por circuitos de aprovação standard.

Toda a informação de catálogo e stock é mantida na plataforma Commerce, que tembém suporta toda a parte de carrinho de compras.

Uma vez que a informação se encontra distribuída nas plataformas Commerce, Sharepoint e ainda em alguns sistemas externos, opta-se pela utilização da componente de indexação para unir esta informação dispersa, através da construção de uma plataforma de indexação costumizada, composta por uma componente de listagem e um segundo nível com o detalhe dos produtos.

Padrão MVC (Model-View-Controller)

Tema para um artigo só por si, o MVC é um padrão de implementação que merece uma breve referência para enquadramento (e para despertar o interesse pelos padrões, para quem ainda não os conhece).

O padrão arquitectural MVC separa as camadas aplicacionais em 3 componentes:

- Model: responsável pela representação e manutenção dos estados dos dados;
- View: camada de apresentação, responsável por mostrar os dados e interagir com o utilizador;
- Controller: camada de lógica que toma as decisões baseada nos dados inseridos pelos utilizadores na View, passando a informação necessária ao Model correspondente para manuseamento dos dados.

Este padrão apresenta como principais vantagens:

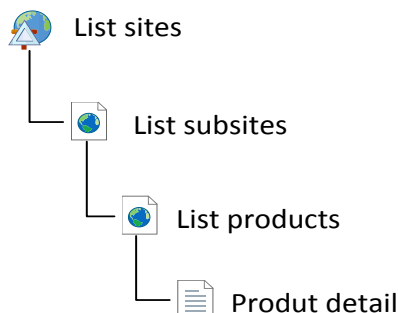
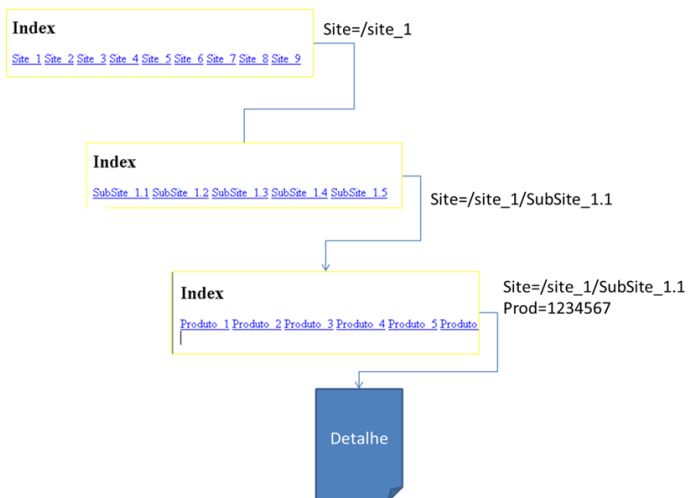
- Melhor organização da aplicação pela divisão das várias lógicas em zonas separadas;
- Não usa view state ou forms, o que dá um maior controlo sobre a aplicação e sobre a informação trocada com o utilizador;
- Melhor suporte para um padrão de desenvolvimento TDD (Test Driven Development é um padrão que promove o desenvolvimento baseado na existência de testes unitários);
- Dada a granularidade fina dos componentes, é uma boa aposta para equipas de maior dimensão.

Indexação e pesquisa

Tendo a pesquisa como fonte de dados da camada de apresentação, vários pontos surgem como requisitos deste componente:

- Fornecer informação de carácter genérico, contida num portal de Sharepoint;
- Fornecer informação de catálogo, contida no Commerce Server;
- Fornecer informação de cariz técnico, contida em outros sistemas externos;

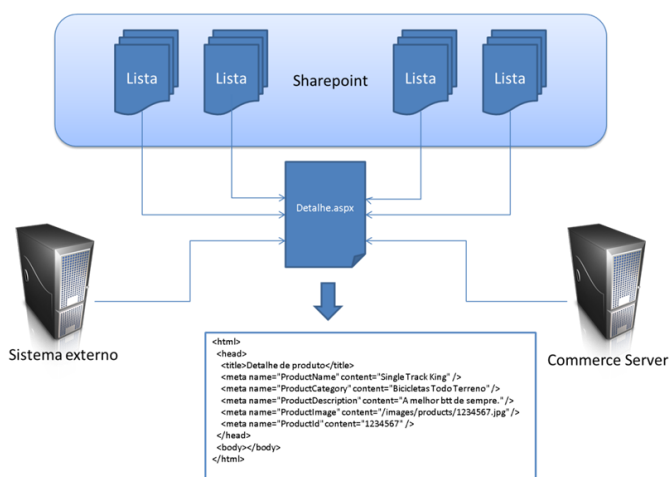
Com base nestas necessidades optou-se pela seguinte abordagem:

- Criação de um conjunto de páginas de índice em cascata, que começam por listar sites com link para o nível seguinte onde passa o site seleccionado no link; no nível seguinte lista os sub-sites do site seleccionado e por último, lista os produtos com link para a página de detalhe do produto.

- Página de detalhe do produto que recebe por parâmetro o sub-site, bem como o código do produto e gera uma página de detalhe contendo os dados necessários para a camada de apresentação. Estes dados, ao serem indexados na mesma "página" ficam relacionados entre si. Ao incluirmos uma chave (ex. id do produto), vamos poder aceder-lhe como uma entidade.

Os principais desafios da componente de indexação, revelam-se nesta altura e vale a pena salientar:

- Uma vez que para um comportamento correcto desta página, os dados têm de ser indexados em metatags e consequentemente são indexados como texto pelo Sharepoint;

- Ao indexar via metatags, perde-se a caracterização dos dados (ver "O que perdemos com a indexação via metatags");

- Esta página tem de aceder a vários sistemas, desde listas de Sharepoint a web services e WCFs de outras plataformas, o que a torna pesada; uma abordagem multithreaded permite paralelizar as várias recolhas e diminuir significativamente o tempo de carregamento da página. Uma vez que esta página será chamada tantas vezes, quantos os produtos existentes, tem um impacto significativo na duração total do processo de indexação e consequentemente na desactualização da informação para o utilizador final (p.e. se uma indexação demorar 1 hora, há a probabilidade dos dados dos produtos apresentados estejam desactualizados até 1 hora atrás).

- O modelo do processo de recolha de informação e geração da página de indexação é apresentado no diagrama seguinte.

O código gerado pela página de detalhe do produto é algo do tipo:

```
<html>
  <head>
    <title>Detalhe de produto</title>

       <meta name="ProductName"
             content="Single Track King" />

       <meta name="ProductCategory"
             content="Bicicletas Todo Terreno" />

       <meta name="ProductDescription"
             content="A melhor btt de sempre." />

       <meta name="ProductImage"
         content="/images/products/1234567.jpg" />

       <meta name="ProductId"
             content="1234567" />
       …
  </head>
  <body></body>

</html>
```

Ao ser indexada esta página fará com que sejam criadas propriedades (crawlled properties de Sharepoint), com os nomes ProductName, ProductCategory, ProductDescription, etc. que por sua vez poderão ser promovidas por configura-

A abordagem do processo de indexação é propositadamente simplista, para permitir tirarmos proveito do modelo multithreaded da pesquisa, levando-nos a construir as páginas tão simples e pequenas quanto possível, evitando desta forma esgotar os recursos das máquinas.

O que perdemos com a indexação via metatags

O processo de indexação baseado em metatags tem uma desvantagem imediata que é a descaracterização do tipo de dados, uma vez que estaremos a indexar páginas HTML, em cujos campos de meta são sempre indexados como texto. No nosso cenário, esta desvantagem é minimizada, uma vez que os dados indexados são utilizados apenas para geração da apresentação, baseando-se os dados e processo de compra na plataforma Commerce.

É sempre possível efectuar transformações de tipo (cast ou parse) que devidamente efectuadas não trazem desvantagens em termo de estabilidade, mas são operações pesadas em termos de ciclos de processamento trazendo uma degradação significativa da performance quando invocadas frequentemente.

```
string myText = "123456";
  int myVal;

  if (!int.TryParse(myText, out myVal))
  {
     // Ocorreu um erro no processo de parse
     // tratar o erro neste ponto
  }
```

Consumo da informação

Estando a informação devidamente indexada pelo Sharepoint, é altura de ser consumida e renderizada nas máquinas responsáveis pela apresentação dos dados. O consumo da informação, efectuado pelos models MVC, poderá ser feito de duas formas:

- Via API (Query Object Model), necessitando de ter o Sharepoint instalado nestas máquinas; solução de melhor performance, mas potencialmente mais onerosa tendo em conta o licenciamento;
- Via serviços (Query Via serviços (Query Web Service); solução mais flexível mas com alguma desvantagem em termos de performance.

Conclusão

Plataformas com a complexidade do Sharepoint são candidatas a implementações inovadoras, uma vez que disponibilizam um conjunto grande de funcionalidades, para os quais nem todos os cenários foram equacionados pelas equipas de produto e/ou arquitectura. Neste artigo é apresentada uma solução diferente das habitualmente recomendas, mas com provas dadas de escalabilidade e fiabilidade para um ambiente empresarial bastante exigente. Espero com este exemplo conseguir despertar em si, o pensamento fora da caixa.

No próximo artigo desta série, será abordada a componente de implementação da solução, com exemplos de código de cada um dos componentes.

Referências

Nota: apesar de estarmos numa publicação em língua portuguesa, as referências apresentadas são na maioria dos casos para as versões inglesas. Isto deve-se somente a, por experiência própria, o conteúdo em inglês ser mais rico e actualizado que nas restantes línguas.

IIS – http://www.iis.net
Microsoft Commerce Server - http://bit.ly/s5Uu2
Microsoft Sharepoint Server - http://bit.ly/qmpwu3
MVC - http://www.asp.net/mvc
SharePoint Search Service Tool - http://bit.ly/ZY9H3
Test Driven Development - http://bit.ly/qAIxha
The Search Developer Story in SharePoint 2010 - Query Interfaces - http://bit.ly/bnaA4G

AUTOR

Rui Melo - rui.melo@microsoft.com

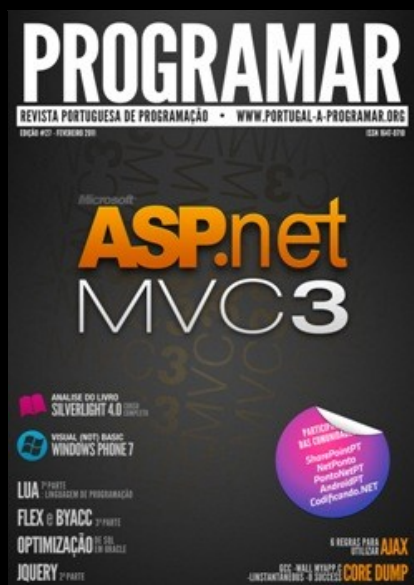
Desde há 12 anos, no mundo das IT, iniciou a carreira na área de I&D de comércio electrónico. Passou por empresas como SMD, Pararede, Newvalue e Link, integrou os quadros da Microsoft como consultor há cerca de 5 anos, mais focado na plataforma Sharepoint. Envolvido desde cedo na comunidade SharepointPT, tem na partilha de conhecimento uma forte motivação. Blog: http://mystepstones.wordpress.com.

Veja também as edições anteriores da Revista PROGRAMAR

30ª Edição - Agosto 2011

29ª Edição - Junho 2011

28ª Edição - Abril 2011

27ª Edição - Fevereiro 2011

26ª Edição - Dezembro 2010

25ª Edição - Setembro 2010

e muito mais em …

www.revista-programar.info

DUVIDAS?

IDEIAS?

AJUDAS?

PROJECTOS?